Inaugural-Dissertation

zur
Erlangung der Doktorwürde

der
Naturwissenschaftlich-Mathematischen Gesamtfakultät

der
Ruprecht-Karls-Universität
Heidelberg

vorgelegt von:	Diplom-Informatiker **Lorenzo Masetti** aus Florenz
Tag der mündlichen Prüfung:	25 Oktober 2011

Implementation of a Large Scale Control System for a High-Energy Physics Detector: The CMS Silicon Strip Tracker

Lorenzo Masetti

Gutachter: Prof. Dr. Volker Lindenstruth

Prof. Dr. Peter Fischer

Implementation of a Large Scale Control System for a High-Energy Physics Detector: The CMS Silicon Strip Tracker

Control systems for modern High-Energy Physics (HEP) detectors are large distributed software systems managing a significant data volume and implementing complex operational procedures. The control software for the LHC experiments at CERN is built on top of a commercial software used in industrial automation. However, HEP specific requirements call for extended functionalities.

This thesis focuses on the design and implementation of the control system for the CMS Silicon Strip Tracker but presents some general strategies that have been applied in other contexts.

Specific design solutions are developed to ensure acceptable response times and to provide the operator with an effective summary of the status of the devices. Detector safety is guaranteed by proper configuration of independent hardware systems. A software protection mechanism is used to avoid the widespread intervention of the hardware safety and to inhibit dangerous commands. A wizard approach allows non expert operators to recover error situations and minimizes the downtime due to errors. Finally, the possibility of using a model-based methodology for prototyping 3D user interfaces for control systems is investigated.

The presented software is in continuous operation for the control of the CMS tracker while some general solutions have been adopted in the control systems of other CMS sub-detectors.

Implementierung eines umfangreichen Kontrollystems für den Siliziumstreifen-Tracker-Detektor des CMS Hochenergiephysik-Experimentes

Kontrollsysteme für Teilchendetektoren moderner Hochenergiephysik-Experimente (HEP) bestehen aus einer Vielzahl von verteilten Softwaresystemen, die eine signifikante Menge an Daten verarbeiten und komplexe Betriebsprozesse steuern. Bei den Experimenten des LHC am CERN wurden die Kontrollsysteme auf das kommerzielle Programmpaket PVSS von ETM aufgebaut, welches auch für die industrielle Automatisierung verwendet wird. Jedoch forderten die speziellen Anforderungen der HEP eine weitreichende Erweiterung der Funktionalität.

Diese Arbeit konzentriert sich auf das Design und die Implementierung des Kontrollsystems für den CMS-Siliziumstreifen-Detektor, zeigt jedoch auch generelle Strategien auf, die in anderen Kontexten erarbeitet wurden.

Es wurden spezifische Lösungen entwickelt, die geringe Reaktionszeiten des Systems gewährleisten und dem Operator effektive Übersichten über den Status des Systems und dessen Bausteine bieten. Die Sicherheit des Detektors wird durch die Konfiguration unabhängiger Hardware-Systeme gewahrt. Ein zusätzlicher, softwarebasierter Schutzmechanismus wird genutzt, um den Eingriff des Hardware-Sicherheitssystems auf ein Minimum zu reduzieren, und um potenziell gefährliche Befehle der Operatoren zu unterdrücken. Des weiteren erlaubt ein "Wizard" auch Operatoren, die keine Experten sind, nach einer Störung wieder zum normalen Betrieb zurück zu gelangen, und somit die Ausfallzeiten zu minimieren. Zuletzt wird die Möglichkeit untersucht, eine modellbasierte Methodik zur Verwendung von dreidimensionalen Benutzeroberflächen für das Kontrollsystem zu verwenden.

Die derzeitige Software wird dauerhaft für die Kontrolle des CMS-Trackers verwendet, wobei einige generelle Lösungsansätze auch für die Kontrolle von anderen CMS-Subdetektoren übernommen wurden.

Contents

Introduction

The new generation of High-Energy Physics (HEP) experiments poses new challenges to the field of control systems. The control system of a large detector, such as the ones installed at the Large Hadron Collider (LHC) at CERN, must evaluate and operate on more than one million parameters. These are read out from heterogeneous front end devices enabling detector operation and ensuring its safety. Despite the complexity of the equipment, the control system should provide the user with a straightforward and effective interface for monitoring and commanding the status of the hardware. In order to cope with the huge data volume and with the complex and varied tasks needed to operate the experiment, the control system must be distributed over several PCs managing autonomous but co-operative entities, hierarchically organized and integrated in an overall supervisor. The control systems for each particular sub-component or sub-detector of the experiment are developed in parallel and finally integrated to ensure coordination between the various components. To facilitate their integration, the various sub-projects are developed following the guidelines and exploiting the functionalities of a common framework developed at CERN.

The present thesis focuses on the design and implementation of the control system for the Silicon Strip Tracker of the CMS experiment. From the point of view of controls, the CMS tracker is one of the most complex LHC sub-detectors. The Tracker Control System manages about 10^5 power supply parameters, 10^4 parameters read out from the Safety System and 10^5 additional environmental values, read out by the Data Acquisition (DAQ) system. The complexity of the control system called for innovative solutions both with respect to the current state of the art, represented by modern industrial automation softwares, and to the general methodologies adopted in the control of the LHC experiments.

Given the dimensions of the system, a particular emphasis is given to performance issues. Possible performance bottlenecks were identified and specific design solutions were established to resolve them. Original strategies include a caching mechanism for most of the static information, maximal role distribution among the various machines in order to

avoid unnecessary data exchange over the network and fine tuning of the hardware drivers. The detector is controlled by a hierarchy of Finite State Machine (FSM) objects reporting the state of the detector at different levels of detail. This standard approach needs to be integrated with quantitative information, providing the user with an effective summary of the status of the individual devices. This requirement led to the development of a general library to propagate the status changes in real time without overloading the system. Given the success of this method, this library was later adopted by other sub-detectors. Moreover, the control system includes a tool that automates the analysis and resolution of problems. This kind of analysis, usually left to human experts, can be effectively performed automatically, thus minimizing downtime and relieving the experts from unnecessary work.

Outline

Chapter 1 presents the general layered and modular architecture of Supervisory Control and Data Acquisition (SCADA) systems, commonly used for the control of industrial facilities and distributed infrastructures.

Chapter 2 gives an overview of the requirements for the CMS Tracker Control System, in the context of the LHC experiments. The detector and the control front end hardware are described underlining the resulting requirements for the control system.

Chapter 3 discusses the specific requirements for a HEP experiment control system. The chapter includes the description of the common framework developed at CERN in order to integrate control-specific tasks with HEP-related features, minimize the duplication of work and ensure the homogeneity of the developments.

Chapter 4 is the core of the thesis. It presents the innovative and original solutions developed for the control of the CMS tracker and the principles that guided the design and the development.

Finally, Chapter 5 presents the results of a study investigating a model-based development for the automatic prototyping of three-dimensional interfaces for complex control systems.

Chapter 1

Large Control Systems: Domain Definition, State of the Art and Trends

The present thesis is focused on the design and implementation of the Detector Control System (DCS) for a High-Energy Physics (HEP) experiment. This type of control software belongs to the general category of Supervisory Control and Data Acquisition (SCADA) systems. This chapter presents the scope and the common problems to be addressed in this type of distributed software. One particular commercial SCADA product was selected for the implementation of the control systems at CERN. Its architecture is presented here in detail, in the context of the general architecture of a SCADA system.

1.1 Definition of SCADA

Supervisory Control and Data Acquisition (SCADA) [1] is a general term for a system for monitoring and controlling a remote process[1]. SCADA systems cover a wide range of applications. Typical fields where SCADA systems are used include:

- industrial processes, such as manufacturing, production, power generation

[1]The meaning of the term *Data Acquisition* in the context of SCADA is different from the meaning that the term has in experimental physics, where it is referred to the data-taking from the equipment for physics purposes. In a SCADA system the term refers to the reading of the parameters that characterize the front end devices and has to deal with a much lower data rate than a Data Acquisition (DAQ) system of a HEP experiment.

- infrastructure processes, such as water treatment, oil and gas pipelines
- facility processes in buildings, airports, ships and space stations. SCADA systems can be used in this kind of environment for heat and ventilation control, access management or energy consumption monitoring.

Control systems are by nature distributed and are typically organized into three layers:

- a *supervisory layer* gathering (acquiring) data from the controlled process for visualization and analysis and sending commands (control) to the process
- the *front end layer*, connected to sensors in the process, converting sensor signals into digital data and sending digital data to the supervisory system
- a *communication layer*, connecting the supervisory layer to the front end layer.

The supervisory layer also implements a Human-Machine Interface (HMI) that enables the operator to visualize the state of the system and to give commands. The front end layer can be geographically distributed over large areas and in some cases (especially in industrial environments) access to the front end devices can be restricted for safety reasons. These conditions call for a stable and reliable control system, allowing the remote operation of the process.

SCADA systems are not a complete control system, but only its software component, focused on the supervisory level, positioned on top of the controlled hardware. Most commercial products used in this context are SCADA toolkits that provide a general framework for building customized SCADA systems.

As the acronym states, a SCADA system implements three distinct functionalities (Supervision, Control and Data Acquisition).

Supervision is related to all the software tasks that enable the observation of the present or historical status of the system, including the processing and analysis of acquired data.

Control tasks imply the change of the state of the system and can be triggered by user commands or by automatic actions, in response to the state of the process.

Data Acquisition tasks are related to communication with the front end devices for reading and writing data. The primary objective of a

SCADA system is not the data acquisition itself, but rather the supervision and control tasks allowed for by the data [2]. Moreover, instead of data acquisition it would be more correct to talk about *data exchange*, since the communication with the front end layer is bidirectional.

Historically a distinction was made between SCADA and Distributed Control Systems (DCS)[2]. This distinction is based on the grade of distribution of the system intelligence. According to the traditional definition, in SCADA systems the elaboration and control functions are all implemented in the supervisory level. On the other hand in DCS systems, the intelligence is distributed over many controllers that implement specific control tasks. The most common example of Distributed Control Systems are control loop systems that attempt to correct the error between a measured process variable and a desired setpoint by calculating and then outputting a corrective action that can adjust the process accordingly. Currently this distinction is disappearing in the terminology and SCADA refers to the high level software component of the control architecture. In most cases a modern control system includes intelligent processing in the front end layer to deal with safety-critical or time-critical tasks.

1.2 The Supervisory Layer: Requirements and Architecture

Basic requirements [3] of a modern SCADA system used in industrial automation are:

- data collecting capabilities to effectively conduct the desired measurement and control tasks
- uninterrupted system monitoring and recording of the gathered information to a historical database
- software versatility allowing the configuration of system parameters and the programming of ad hoc supervising procedures
- possibility to develop a user-friendly Graphical User Interface (GUI) providing a convenient way of interacting with the system
- comprehensive alerting and reporting capabilities in order to promptly inform the personnel in the presence of emergent situations

[2]The term DCS is used here with the general meaning of Distributed Control System, different from the meaning of Detector Control System, that is used throughout the full document.

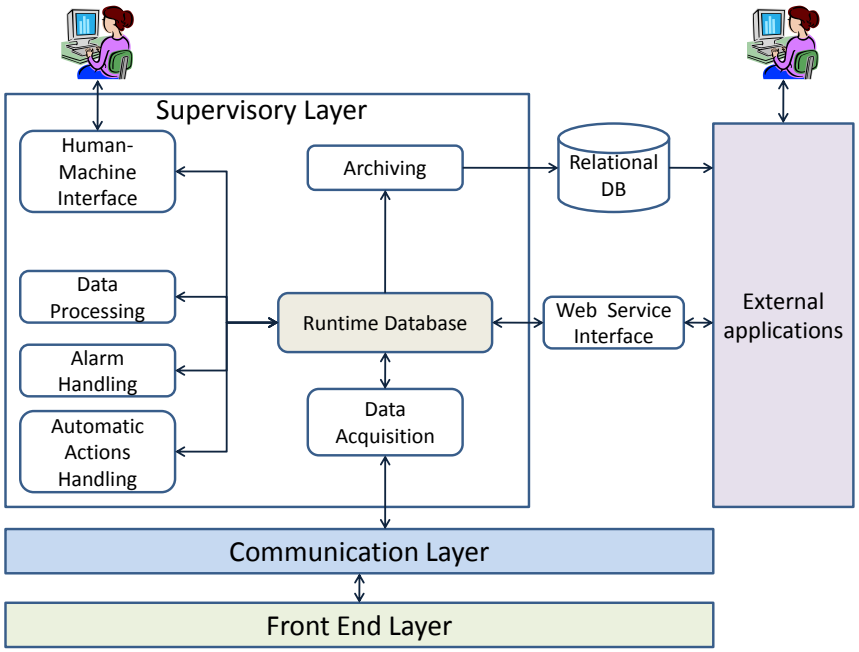

Figure 1.1: Architecture of the supervisory layer of a SCADA system. A centralized collector (*runtime database*) provides access to the data. The various functional modules communicate with the Runtime Database only. The users interact with the HMI layer of the SCADA system and with external applications. External applications can access historical data via the relational database and online data via a web service interface.

- openness to the external world by means of standard interfaces.

The supervisory layer of a SCADA system must implement three fundamental functionalities: data collecting, data processing and publishing of the information to external systems.

Data collecting requires a centralized access point where the system image (i.e., the values of all the variables describing the status of the controlled process) is stored, continuously updated and made available to all the software processes needing access to the data (see Fig. 1.1). This centralized collector, usually termed *runtime database*, should provide the typical database features and must ensure a very short access time. For every data element, the runtime database must provide at least one data identifier (typically a string), the timestamp of the last reading, the current value and a quality flag. For better performance, the system image is typically kept in memory. Data elements are usually called *tags* or *points*. A distinction can be made between hard and soft points. A hard point represents an actual input or output value, monitored or controlled

by the SCADA system, while the value of a soft point is computed from the values of other points. Most implementations treat hard and soft points in the same way, allowing the developer to define which points correspond to a specific hardware address.

Since the controlled hardware is often very expensive and has a long lifetime, SCADA system are typically operated for years without significant changes to the software. The architecture of SCADA products meets these requirements. The runtime database is designed to be quite static. It is typically configured once during deployment and then used to operate the hardware for a long time without changing its structure.

Due to the processing model of the SCADA system, the natural way to communicate between processes is an event-driven architecture, where the computation flow is determined by events. In this case, events correspond to changes in the value of the points in the runtime database. This is preferred over an architecture where the clients contact the runtime database periodically (polling) because in case of stable processes the system load is much lower with the event-driven approach.

The *smoothing* mechanism is the foundation of an event driven system. In its most simple form (old/new comparison) it compares the new value with the previous value of the element, generating an event only if the new value is different. In case of measured values, it is useful to define a threshold (deadband), related to the precision of the instrumentation, and to update the value only when the difference with the previous value is greater than the defined deadband.

The acquired data has to be archived in order to analyze the evolution of the process. The recommended tool to archive historical data is a relational database. Relational databases provide a standard interface to external applications and are suited to handle large amounts of data. They have a longer response-time compared to the runtime database, but since the archived data is accessed for offline analysis and not for control purposes, longer access times are not an issue. The event-driven strategy can also be used for archiving. In this case the deadband is typically larger than the one used for the data acquisition. This allows the control system to report small data fluctuations to the user while only significant trends are archived into the database.

Given the high data volume and the great complexity of the controlled process, a human operator cannot analyze all the collected data. For this reason a SCADA system must implement real time procedures to generate high level information presented to the user. Regarding commands, the user should be provided with high level commands instead of having to control each specific device individually. To fulfill these requirements, data processing must provide the means to deduce the relevant summa-

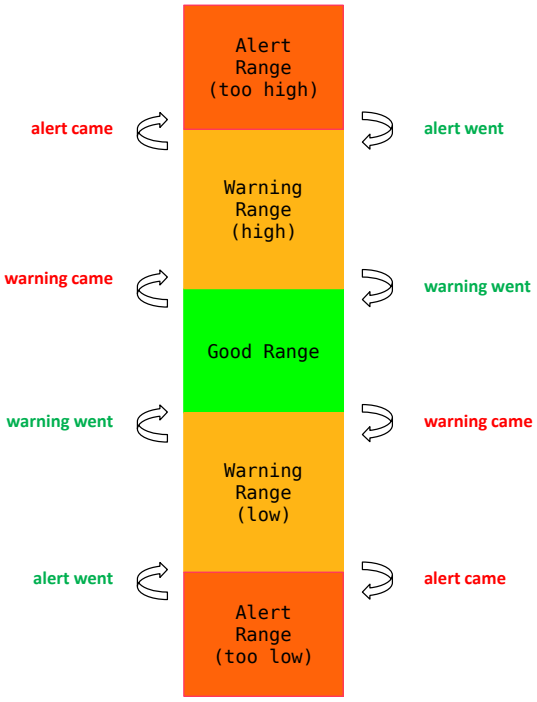

Figure 1.2: Example of alert ranges and possible transitions. The range of all possible values of an element is partitioned into regions corresponding to warning or critical conditions. CAME transitions occur when the value moves to a more severe alert range. WENT transitions occur when the value moves to a less severe alert range.

rized information from the primary data read from the devices, signal to the user any anomalous value, translate high level commands into proper sequences of low level actions and provide automatic control procedures in response to certain state changes.

To achieve the needed flexibility in data processing, SCADA programmers should not be limited to the configuration of the runtime database or to the design of the user interface. Instead, the SCADA toolkit must be customizable with the help of a general-purpose programming language, fully integrated with the SCADA functions.

An important feature of a SCADA system is alert handling. Alerts are used to identify anomalous values in the system that should be reported to the user who can take some corrective actions. The range of all possible values of a parameter can be partitioned into two or more alert ranges. Each alert range has a priority (0 for the normal range). A CAME transition is a transition into an alert range with higher priority while a WENT transition is a transition to an alert range with lower priority (see Fig. 1.2). Alerts must typically be acknowledged by the user. If

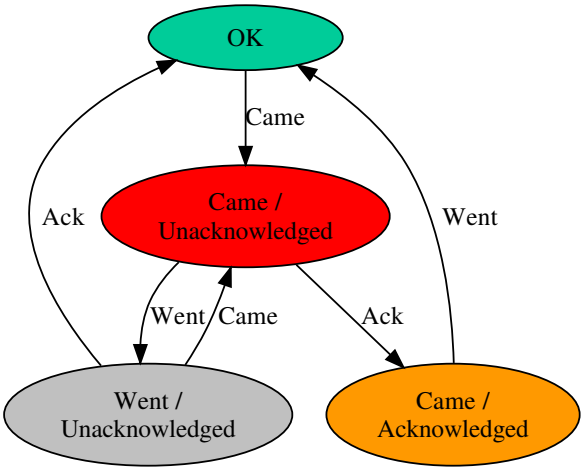

Figure 1.3: Possible states and transitions for an acknowledgeable alert. After a critical situation, the alert state can go back to OK only if the user has acknowledged the presence of the alert condition.

the alert has to be acknowledged and the condition passes before the user acknowledgement, the alert goes to state WENT/Unacknowledged, so that the information regarding a past critical situation is not lost (see Fig. 1.3).

Finally a SCADA system must provide the tools for *exiting the system* by publishing the data in standard open formats. Historical data are archived in a relational database that provides a standard interface for web tools and advanced visualization software. External applications performing specialized tasks may also require access to online data. This type of communication is typically provided via a web-service interface that uses a service-oriented architecture. The use of a standard protocol favors the integration with third-party software components. The use of external specialized applications is desirable since they can be specially designed to handle additional functionalities while allowing the SCADA system to focus on control tasks.

1.3 The Front End Layer

The front end layer is directly connected to the controlled hardware and performs the first processing steps. Front end layer devices can be used to measure various physical properties (temperature, humidity, voltages, currents) or to provide specific services (power system, pumps, air conditioning, cooling). Digitization is typically performed at the front end layer.

Programmable Logic Controllers (PLC) are highly robust diskless digital computers that are commonly used at the front end layer to implement real-time control features. PLCs produce output results in response to input conditions within a bounded time. Inputs are collected from hardwired sensors measuring analog process variables, such as temperature and pressure, or from complex positioning systems. Outputs are connected to the *actuators* and can be used to operate electric motors, pneumatic or hydraulic cylinders, magnetic relays and so on.

1.4 The Communication Layer

The current trend in the field control systems is the standardization of the communication between the front end and supervisory layers. Standard network protocols used in the Internet, such as TCP/IP and UDP/IP over Ethernet are commonly adopted. This choice enables use of inexpensive components, existing infrastructure and well-established debugging techniques.

At the application level, OPC (OLE for Process Control) [4] is emerging as a *de facto* standard for process control, providing a homogeneous way of communicating with different hardware devices. Each hardware manufacturer typically provides a specific OPC server which acts as a protocol converter for communicating with its custom hardware devices. The main advantage of standardization at the application level is that the communication with the hardware can be established with some configuration effort rather than with low-level driver programming.

The OPC standard was developed in 1996 in order to solve the lack of standardization in the field of inter-connectivity to different hardware. The OPC specification is based on protocols primarily based on Microsoft technologies (currently DCOM) and can only run under the various versions of the Microsoft Windows operating system[3].

While many PLC manufacturers provide an OPC server to communicate with their hardware, other protocols are still commonly used to interface SCADA systems to PLCs, without using a specific software component in the communication layer. Examples of proprietary protocols for communicating with PLCs are Siemens S7, published by Siemens, and Modbus, published by Modicon.

Not all data needed for supervision purposes is directly acquired from

[3]The first step towards a platform-independent implementation of the OPC protocol has been taken with the specification of the OPC Unified Architecture (OPCUA) which is independent from DCOM and is implemented in Java, Microsoft .NET and ANSI C.

front end hardware. In case of large infrastructures, some data acquired in one station must be broadcasted to a significant number of other locations. In this case it is often convenient to use a high level exchange protocol, such as DIM (developed at CERN). Moreover the SCADA system may also need to treat data acquired by an external specialized application. For this purpose external applications can communicate with the SCADA system via a web-interface using a standard protocol. SOAP is a common choice for this type of communication, since it is supported by most programming frameworks.

1.5 Automatic Actions in a SCADA System

Especially for safety critical tasks, experience shows that user-intervention in control systems should be minimized. This recommendation is not based on prejudices or a lack of confidence in the operators, but rather on the negative impact of psychological pressure on operator abilities.

Automation is absolutely needed for safety related actions and time critical tasks. In these cases it is necessary not only to automate the control but also to distribute the safety logic in peripheral devices at the front end level. This ensures high reliability and speed. PLCs are the industrial standard for establishing such safety procedures.

Automatic control procedures are also highly desirable when the decision process is very complex. If a large number of variables needs to be checked in order to determine the appropriate action, a human operator is unable to perform this analysis on an acceptable timescale. The control procedures can be automated in these cases and user interaction limited to the confirmation of some actions.

1.6 Human-Machine Interface Principles

Interaction with the user is carried out by the HMI component of a SCADA System. Usability of the interface is a key-point in the development of a control system. Ideally, the control application should act as a transparent interface between the two real actors, the operator and the controlled industrial process.

A usable and responsive interface increases the confidence of the operators in the system, even in critical conditions, limiting the situations of psychological stress. Minimization of the interaction, automation of repetitive operations, simplification of common user tasks are all elements that contribute to the success of a SCADA system.

Two types of functionalities can be distinguished in the interaction with a SCADA system: data presentation and command handling.

Data presentation typically makes use of schematic representations, symbolical states, tables and temporal trends.

Schematic representation uses standard symbols to improve the usability of the interface. For example, a green LED representing an ON channel is more effective than a text field with the string "ON". The usage of a standard color convention favors an immediate understanding of data presented in symbolic or numerical form. Tables are useful for expert users, especially when a comparison must be made between two or more values. History trends are very effective for the representation of the evolution of the process. Expert operators can easily recognize known patterns from the trends of key variables.

Command submission should be protected from unauthorized access and from accidental undesired actions. It is important to introduce some form of feedback when a command is executed, such as command acknowledgement when the feedback time for the expected state change is longer than a few seconds. In case of complex procedures started by the user and automatically handled by the system, it is important to have a feedback for each step and to show the progress of the operation, to confirm that the system is properly reacting to commands.

Human interaction in modern SCADA systems is not limited to the operators that are physically present in the control room. In SCADA softwares many independent user-interface clients can typically be connected to the system core. The user interface is not limited to standard computer displays but can involve other telecommunication devices. Automatic notifications using SMS or email can also report unexpected events to experts. Moreover, information can be published on the web where it can be accessed from mobile phones or palmtop computers thus facilitating access to SCADA information from outside the control room.

1.7 Security Risks in SCADA Systems

With the adoption of standard communication infrastructures and commercial products, new vulnerabilities are also inherited in the control systems [5, 6]. Security threats are becoming more worrisome because of the desired connection between the dedicated controls networks and the general network.

The security risk can be expressed as a product of three factors: *threat*, *vulnerability* and *consequence*.

Threats originate from worms, viruses and malicious attackers com-

ing from the external network, but also from operators misconfiguring a device or from broken devices that flood the network. Protective measures must be put in place to prevent external threats penetrating control systems and also to prevent the insiders from (deliberate or accidental) unauthorized access.

Inherent vulnerabilities affect most of the actors in the control system networks. PLCs and other controls devices are nowadays directly connected to the Ethernet but often lack security protections. Control PCs cannot be patched as fast as office PCs, as any interruption of the services they provide has to be scheduled beforehand. In addition the control software might not be compliant with some patches. Some protocols commonly used in automation systems, such as OPC, also lack fundamental access control features.

However, these concerns must be addresses since the consequences from suffering a security incident can be very severe. In general SCADA systems are vital components of many critical infrastructures, such as water and transportation systems, chemical plants, power stations. There is a potential for great harm to human life and to the equipment safety if these control systems are breached. In the HEP environment, an incident can lead to damage of unique equipment.

A key point for improving SCADA security is network segregation. Communication between the controls network and the Internet, though unavoidable, should be reduced to minimum. Additional network segregation allows for further protection of vulnerable devices such as PLCs. PLCs and other front end devices accessible via Ethernet should be connected only to the restricted set of PCs that are entitled to communicate with them.

1.8 Evolution of SCADA Systems

The evolution of SCADA systems in the last years and the current trends for future SCADA applications can be summarized in the following points:

Usage of Standard Protocols In the last ten years, the communication layer of SCADA systems moved from proprietary protocols supporting the hardware of a particular vendor, to open standard protocols such as Modbus or OPC over TCP/IP. Open architecture SCADA systems enable users to integrate transparently different manufacturers' equipment.

Openness to the External World Modern SCADA systems are becoming more and more integrated with external applications, pro-

viding a service oriented interface using standard protocols. Using network technologies, a SCADA system is no longer confined to a stand-alone or dedicated network but rather allows operators anywhere to access the data in real-time.

Security Issues Communication with the external world calls for improved SCADA security. Security is traditionally a weak aspect of SCADA systems. Modern SCADA systems should evolve towards more security.

Distribution and Interconnection The supervisory layer architecture relies on a central element (the runtime database) that coordinates all other processes. However, in order to cope with a very large amount of data, the system needs to be distributed over many central coordinators running in several PCs. The distributed elements should be connected to form a network of autonomous and cooperating nodes, exchanging the needed information but still performing most of the work locally.

Object-Oriented Architecture Traditional SCADA products are tag-based. The runtime database provides a completely flat namespace that collects a non structured set of elements. Instead, in object-oriented SCADA systems, each elementary device is controlled within a specific class that provides both the data structure and the relevant methods to operate on that type of objects. Inheritance and polymorphism allow for the definition of standard templates for similar devices, that can then be specialized to implement a specific device model. The object-oriented methodology favors reusability, eases code development and increases the flexibility of the application. However, most SCADA products will most likely not move to a real object-oriented methodology in a near future, because this would imply a major change in architecture. Some leading companies in the SCADA field, such as Wonderware, have already developed SCADA toolkits that use an object-oriented methodology. Wonderware advertises the positive impact of the methodology on the development in terms of lifecycle savings [7]. In any case, most of the aforementioned advantages can also be obtained by structuring the runtime database in types (without object-oriented features).

1.9 Short Survey of SCADA Products

Most SCADA toolkits used in the industrial environment are commercial products. A large number of SCADA products is available on the market. They have varying features and complexity and are aimed at different applications. Examples of widely-used commercial SCADA products are Wonderware InTouch, Intellution iFix, Siemens WinCC and ETM PVSS-II.

In the scientific environment, LabVIEW by National Instruments is a popular choice for controlling small-scale setups. It provides libraries for data acquisition, signal generation and processing. However, the dataflow programming approach is unfit to manage a large number of devices in a distributed system.

Experimental Physics and Industrial Control System (EPICS) [4] is a collaborative and open source set of software tools and applications used in building distributed control systems. It has been successfully used in many HEP applications. EPICS is based on a protocol (Channel Access) that allows different clients to access the points (termed Process Variables) published by different servers in a distributed environment. Different runtime databases (termed Channel Access Services) can be accessed transparently. The client only needs to know the name of the Process Variable, and not the address of the Channel Access Service that publishes it. EPICS can also be used to program closed-loop controllers at the front end level.

CERN chose the application to use for implementing the control systems for the new LHC experiments at the end of the 90's. The specific requirements for HEP experiments leading to this choice will be discussed in detail in Chapter 3.

A survey of the SCADA market was performed, resulting in the selection of PVSS-II[5], a product by the Austrian company ETM (recently absorbed by Siemens). A discussion of the criteria leading to this choice can be found in Section 3.3.

1.10 The PVSS-II SCADA System

PVSS-II implements the general architecture of a SCADA system. It presents many advantages compared to other SCADA products, notably a structured runtime database, a modular architecture and a good scal-

[4] http://www.aps.anl.gov/epics/

[5] Acronym for *Prozessvisualisierungs und Steuerungssystem*, Process Visualization and Control System

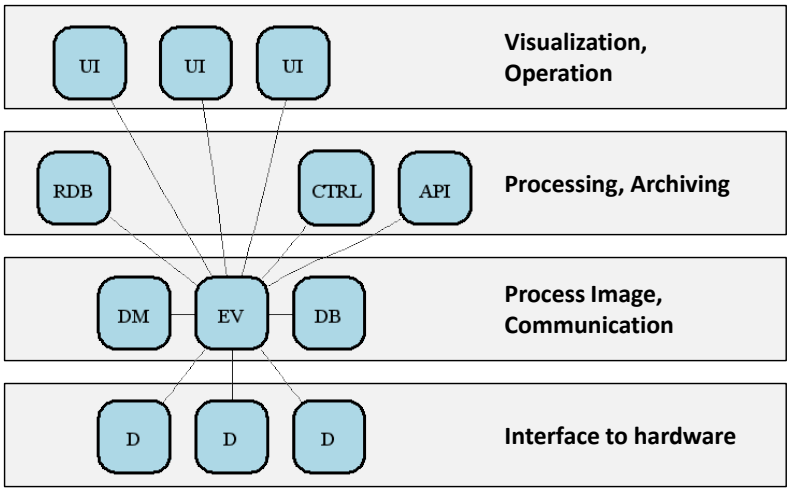

Figure 1.4: Example of the modular architecture of one PVSS-II system containing many independent distributed managers. The Event Manager (EV) keeps the system image in memory. The hardware devices are accessed via dedicated driver (D) managers. Processing, archiving and visualization tasks are carried out by specific managers.

ability thanks to the possibility of interconnecting independent projects.

PVSS is a modular distributed software where the functionalities are implemented in independent processes, called *managers*, that communicate via the standard TCP/IP protocol, either locally or over a local network (see Fig. 1.4). Managers exchange data following an event-driven protocol so that in steady-state operation, when the values are not changing, there is neither communication nor processing load.

Communication with hardware is implemented in special managers called *drivers* (D). The common drivers that are provided with PVSS-II are OPC, S7, ProfiBUS, CANbus, Modbus, TCP/IP and Applicom. The drivers for specific CERN protocols, such as DIM and DIP, have also been developed.

The central processing unit in PVSS-II is the *Event Manager* (EV). The EV keeps in memory the current system image (the runtime database) and ensures the distribution of data to the other managers that request it or that have subscribed to it. Every other manager wanting to access the online data communicates only with the event manager and never with the specific driver.

The runtime database is structured in typed data points, containing a structured set of elements that can be configured to be connected to a particular driver. The managers can interact with the Event Manager by getting or setting the value of a specific element or by subscribing to

its changes.

The *Database Manager* (DB) maintains a persistent image of the status of the runtime database on disk, which is accessed whenever the project is restarted, in order to restore the latest recorded values.

The *User Interface Managers* (UI) provide the HMI for PVSS, while the *Control Managers* (CTRL) are used to run background scripts. Scripts and user interfaces are programmed in a specific interpreted language that uses a C-like syntax but provides some PVSS-specific functions.

PVSS-II is extensible by means of an Application Programming Interface (API) in the form of C++ classes. The developer can implement custom functions as additional independent managers.

A *Relational DataBase (RDB) manager* provides the connection to a relational database for long-term archiving of the monitored parameters. This manager was customized and optimized by the IT department at CERN (see Sec. 3.4.4).

The managers can connect to remote EV and DB managers. If some managers are connected to a remote event manager, running on a different PC, the PVSS system is said to be *scattered*.

Many PVSS systems can be connected by adding a *Distribution Manager* (DM) to each system. Each system keeps a separate namespace for the data (handled in its own EV), but when two systems are connected, the remote namespace is available by adding the system name prefix. To access the data in a remote PVSS system, a manager issues the request to the specific event manager. The communication with the EV of a remote system does not have any specific overhead since it follows the same protocol used for contacting the local EV.

Chapter 2

Requirements for the CMS Tracker Control System

The newest particle physics experiments also pose new challenges to disciplines outside of physics, such as engineering and computer science. This chapter focuses on the control requirements for operating the CERN accelerator complex and the related experiments.

In particular, the working principle and the geometry of the CMS Silicon Strip tracker are presented in this chapter. The front end hardware used for the operation of the detector is described in detail. The Detector Control System (DCS) provides the means to operate the front end hardware remotely and to monitor the environmental conditions of the detector. It plays a pivotal role in ensuring the proper configuration of the safety features programmed at the front end level. Many parts of the controlled hardware are not accessible during LHC operation because of the hostile environment. Finally, the data volume and the expected change rate to be handled in the Tracker Control System is presented.

2.1 The CMS Experiment at LHC

2.1.1 CERN and High-Energy Physics

Particle physics is the branch of physics that studies the elementary constituents of matter and the interactions between them. Modern particle physics research is focused on subatomic particles. Many elementary particles cannot be directly observed under normal circumstances in nature, but can be created and detected during energetic collisions of other particles. The experiments that enable the observation of elementary particles rely on sophisticated detectors that employ a range of advanced technologies to measure and record particle properties. The cost and

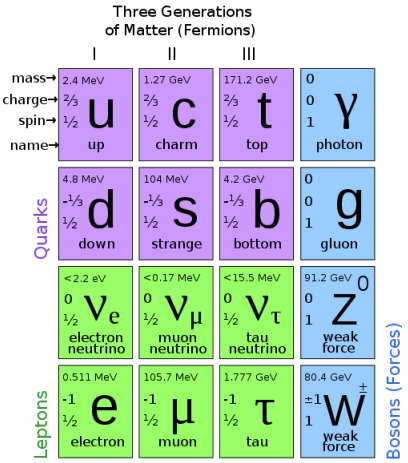

Figure 2.1: The Standard Model of elementary particles, with the gauge bosons in the rightmost column.

complexity of the experiments, accelerators and infrastructure call for international laboratories where large collaborations of scientists work together. CERN, the European Organization for Nuclear Research, is the world's largest particle physics laboratory, established in 1954. The detectors at the LHC accelerator are currently the most advanced experiments in the field of particle physics.

Interactions between elementary particles are currently best described by the Standard Model of particle physics (SM). According to the SM, elementary particles can be divided into fermions and bosons (see Fig. 2.1). Fermions are the particles that constitute matter and are subdivided into two types, leptons and quarks. Leptons include the electron, the muon, the tau and their associated neutrinos, forming three pairings, of increasing mass. Unlike leptons, quarks exist in nature only in combination with other quarks forming hadrons. As in the case of leptons, quarks are organized in pairs of increasing mass. In addition to electric charge, the quarks carry a different kind of charge known as color that is related to the strong interaction.

Each of the pairings fits together in a structure called a generation (see Fig. 2.1) The charged particles of the first generation are the lightest and hence do not decay. Almost all ordinary matter is made of such particles. The charged particles of the second and third generations are more massive and have short lifetimes. These particles are only observed in high-energy environments. Neutrinos are extremely light neutral particles that interact rarely. In addition, for every elementary particle discussed above there is an antiparticle, that is, a particle with the same

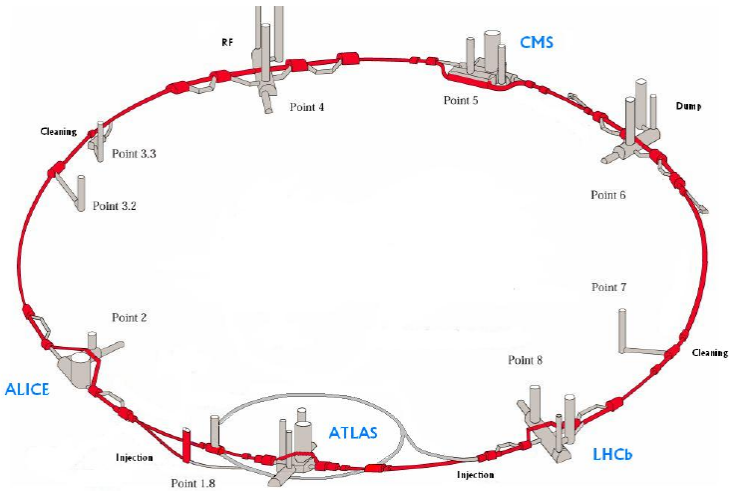

Figure 2.2: Map of LHC underground areas hosting the accelerator, four main experiments and service accesses.

properties but with opposite charge.

Particles interact via three fundamental forces: gravity, the strong force and the electroweak force. Gravity is too weak to have any effect at the elementary particle scale and is not included the Standard Model. In the SM, interactions correspond to the exchange of force mediating particles, called bosons.

The SM has been very successful in explaining experimental results, but cannot be considered a complete theory of fundamental interactions. All the particles in the theory have been experimentally observed and their properties have been studied extensively. There is one exception, the Higgs boson, a particle predicted by the SM, but never confirmed in experiment. Data collected at Large Electron-Positron Collider (LEP) fixed a lower limit for the Higgs boson mass of $\sim$ 114 GeV. The experiments at the new Large Hadron Collider at CERN will be able to explore the mass interval up to 1 TeV, an upper limit imposed by the theoretical consistency of the Standard Model.

2.1.2 The Large Hadron Collider

The Large Hadron Collider (LHC) [8] will be the most powerful hadron collider running in the next years. It is built in place of the former Large Electron-Positron Collider (LEP). The LHC tunnel is about 27 km long, located about 100 m underground underneath the French-Swiss border between the Jura Mountains and the Lake Geneva. It will provide two proton beams colliding at a center of mass energy of 14 TeV. The beams

are initially accelerated by other CERN facilities: a linear accelerator, a Booster, the Proton Synchrotron (PS) circular accelerator and the Super Proton Synchrotron (SPS) that injects 450 GeV proton beams into the LHC ring. As the collisions occur between particles of the same kind, superconducting magnets generate a vertical magnetic field with opposite direction in the two beam pipes.

The beams are brought to collide in four experimental areas where the four main LHC experiments are located: ATLAS at Point 1, ALICE at Point 2, CMS at Point 5 and LHCb at Point 8 (see Fig. 2.2). ATLAS and CMS are multi-purpose experiments, while the other two are dedicated experiments, one to heavy ion physics, ALICE, and the other to b quark physics and precision measurements of CP violation, LHCb.

As the particles of interest have very low production cross sections, in order to collect significant statistics, the rate of collisions has to be higher than in any earlier experiment. The parameter that is commonly used to quantify the rate of the collisions is the *luminosity* that is proportional to the number of protons in the two colliding bunches, to the revolution frequency of the bunches, and to the number of bunches, and inversely proportional to the cross section of the beam. To achieve a high luminosity, the two beams will contain 2 808 closely-spaced bunches filled with an average of 1.15×10^{11} protons. The bunches collide every 25 ns, corresponding to a frequency of 40 MHz.

2.1.3 Overview of the CMS Experiment

The Compact Muon Solenoid (CMS) experiment [9] is a general purpose detector operating at the LHC. CMS is built following the typical structure of a general purpose detector designed for a collider: several cylindrical layers coaxial to the beam direction, referred to as *barrel* layers, closed at both ends by detector disks orthogonal to the beam pipe, the *end caps*. The detector has a full length of 21.6 m, a diameter of 15 m and a total weight of 12 500 tonnes (see Fig. 2.3).

The coordinate frame used to describe the detector geometry is a right-handed cartesian system with the x axis pointing to the center of the LHC ring, the z axis coincident with the CMS cylinder axis (parallel to the beam direction) and the y axis directed upwards, orthogonal to the beam direction. Because of the cylindrical symmetry of CMS design, it is convenient to use an alternative pseudo-angular reference frame, given by the triplet (r, φ, η) where r is the distance from z axis, φ the azimuthal coordinate with respect to x axis and η is the pseudorapidity defined as $\eta = -\ln(\tan(\theta/2))$, θ being the angle from the positive z semiaxis.

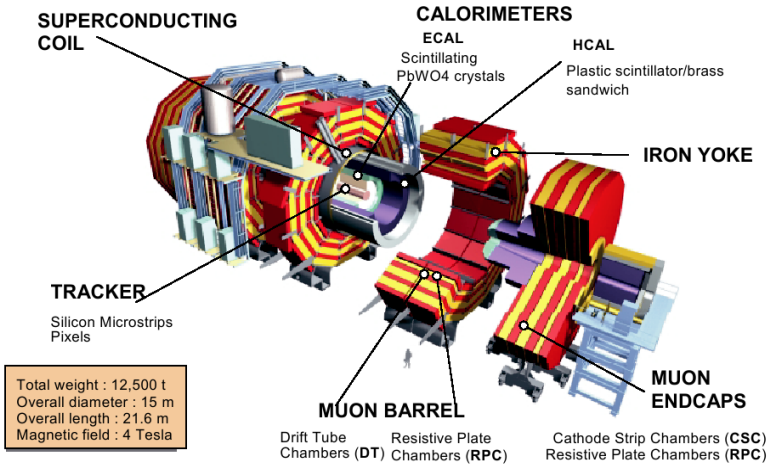

Figure 2.3: Schematic picture of the CMS experiment at LHC. CMS is composed of various sub-detectors that enable detection and measurement of the energy of the different particles that are produced in a collision.

In this reference frame, we can describe the CMS sub-detectors, installed radially from inside out:

Tracker $r < 1.2$ m $|\eta| < 2.5$ Silicon tracker composed of a pixel vertex detector and an outer Silicon Strip Tracker (SST) to reconstruct charged particle tracks and determine primary and secondary vertices. The pixel detector is composed of 3 barrel layers and 2 disks per side. The SST is composed of 10 layers in the barrel and 3 inner plus 9 outer end cap disks per side.

ECAL 1.2 m$< r < 1.8$ m $|\eta| < 3$ Electromagnetic calorimeter designed to measure with high accuracy the energies of electrons and photons, composed of $PbWO_4$ scintillating crystals and a forward preshower detector.

HCAL 1.8 m$< r < 2.9$ m $|\eta| < 5$ Hadron calorimeter system for jet position and transverse energy measurements, extended in the forward region with a very forward calorimeter.

Solenoid Magnet 2.9 m$< r < 3.8$ m $|\eta| < 1.5$ The CMS magnet is a 13 m long superconducting solenoid with a diameter of 5.9 m. It is large enough to accommodate most of the calorimeters and the inner tracker. It provides an inner uniform 4 T magnetic field, which allows a precise measurement of the transverse momentum of charged particles, determined from the curved track that the particle follows in the magnetic field.

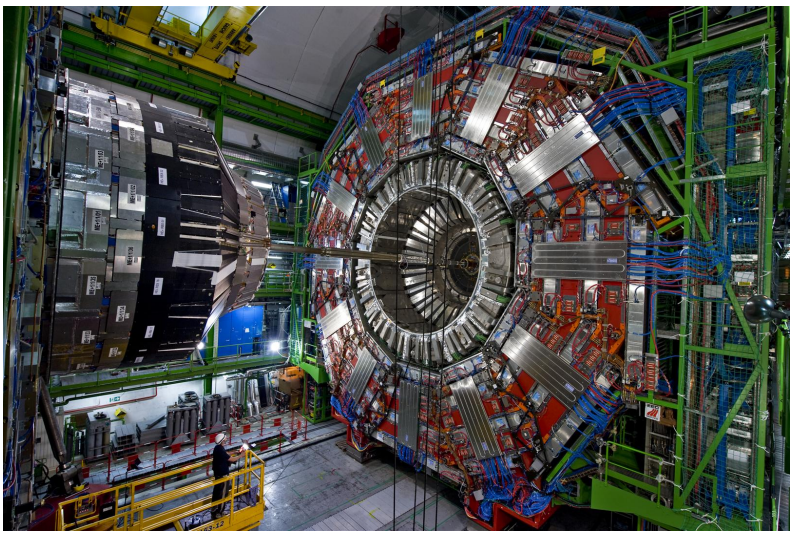

Figure 2.4: View of the CMS detector before closure (August 2008)

Muon System **4.0 m$< r <$ 7.4 m** **$|\eta| <$ 2.4** Muons provide a signature of most of the physics processes the LHC is designed to explore. Fast muon identification and precise measurements of the transverse momentum are thus essential for the LHC experiments. Three types of detector are used in the CMS Muon System: Drift Tubes (DT), Cathode Strip Chambers (CSC) and Resistive Plate Chambers (RPC). The DTs are used for precise trajectory measurements in the central barrel region, while the CSCs are used in the end caps. The RPCs provide a fast signal when a muon passes through the muon detector and are installed in both the barrel and the end caps.

To have a good chance of producing a rare particle, such as a Higgs boson, a very large number of collisions are required. The amount of raw data from each crossing is approximately 1 MB, which at the 40 MHz crossing rate would result in 40 TB of data a second, an amount that the experiment cannot process or store. The CMS Trigger and Data Acquisition System (TriDAS) is designed to inspect part of the detector information at the full crossing frequency and operate a drastic selection, reducing the event rate to a manageable value of 100 Hz. The rate of interesting physics events is orders of magnitude smaller than the total interaction rate. The TriDAS system must be able to reject a factor of 4×10^5 of the collisions while ensuring the selection of interesting events with high efficiency.

Contrary to ATLAS and other large general-purpose experiments, in

CMS the selection task (the Trigger) is split into just two steps. The first step, the Level-1 Trigger, is designed to reduce the rate of events accepted for further processing from 40 MHz to less than 100 kHz. It is implemented in fast hardware logical circuits and is based exclusively on calorimeter and muon chamber information. The remaining event selection, the High Level Trigger (HLT), is implemented in an analysis software running on a commercial computer farm.

The CMS experiment is installed at Access Point 5 of the LHC ring, near Cessy. The experiment is located in the Underground eXperimental Cavern (UXC) that will be a hostile environment with high radiation levels and a strong magnetic field during LHC operation. All the electronics located in UXC have to be radiation hard, while ordinary electronic equipment can be used in the adjacent Underground Service Cavern (USC) in a standard environment. The hostile environment of the CMS cavern during operation of the accelerator calls for sophisticated and reliable remote and automatic control of the devices placed in the UXC.

After the online data taking phase, data has to be stored and analyzed offline. Computing and storage requirements for CMS and the other LHC experiments would be difficult to fulfill at any one place, for both technical and funding reasons. Therefore, the LHC computing environment uses Grid services, with distributed computing and storage resources, available to all collaborators, independently of their physical locations and on a fair share basis.

2.2 The CMS Silicon Strip Tracker

2.2.1 Silicon Microstrip Detectors

The electrical and physical properties of silicon derive from its crystal structure. In silicon the energetic levels of the outermost atomic electrons are distributed in bands of closely spaced energy states separated by forbidden energy regions [10]. The higher energy band is called *conduction* band and corresponds to those electrons that are free to migrate through the crystal. The next lower-lying band, called the *valence* band, represents electrons that are bound to specific lattice sites within the crystal. The separation between the energy of the lowest conduction level and the highest valence one is referred to as the *bandgap* E_g. This fundamental parameter determines the energy needed to excite an electron up to the conduction band, leaving at the same time an empty level in an otherwise filled valence band. This vacancy can be filled by an electron from an adjacent bond, creating a new vacancy in another position.

The moving vacancy can be treated as a positive charge free to move in the material and is called a *hole*.

In silicon E_g is equal to 1.12 eV (at room temperature) and this relatively low value classifies this material as a semiconductor. The small size of the forbidden gap makes silicon an attractive material for detecting the passage of particles.

When there are no impurities in the crystal, the material is said to be intrinsic and the densities of electrons and holes are equal. In the case of silicon detectors, it is convenient to introduce in the crystal appropriate impurities in minute but well-controlled amounts. This process is called *impurity doping* and the material that results is termed a *doped semiconductor*.

Silicon has four electrons in the valence shell, each one makes a covalent bond with one electron of a neighboring atom. The resulting structure is a regular repetition in three dimensions of a unit cell having the form of a tetrahedron with an atom at each vertex.

If some of the silicon atoms are replaced by pentavalent ones like phosphorus, four of the five outer shell electrons will form covalent bonds; it then takes very little thermal energy to free the extra electron for conduction. At room temperature, essentially every pentavalent atom contributes an electron for conduction. This type of elements donate excess electron carriers and consequently are called donors or n-type impurities. Each donor atom that becomes ionized by giving up an electron will leave a fixed positively charged ion in the crystal lattice.

Instead, when trivalent atoms like boron are added to the silicon, only three of the silicon covalent bonds can be filled; the vacancy existing in the fourth unsaturated bond can easily trap a nearby electron from silicon atoms resulting in a new adjacent vacancy. This mobile vacancy represents a hole that can move through the lattice. Such elements are known as acceptor or p-type impurities.

The concentrations of donor and acceptor atom are referred to as the *donor impurity concentration* N_D and the *acceptor impurity concentration* N_A. A material where $N_D > N_A$ is referred to as n-type material, while in a p-type material $N_A > N_D$.

In a single semiconductor crystal, donor impurities can be introduced in one side and acceptors on another side, forming a *p-n junction*. Because of the density gradient across the junction, holes will initially diffuse towards the n-type region and electrons will start to migrate in the opposite direction. The overall result is that the positive holes that neutralized the acceptor ions near the junction in the p-type silicon have recombined with electrons that have diffused across the junction. Similarly, the free electrons in the n-type silicon have recombined with holes

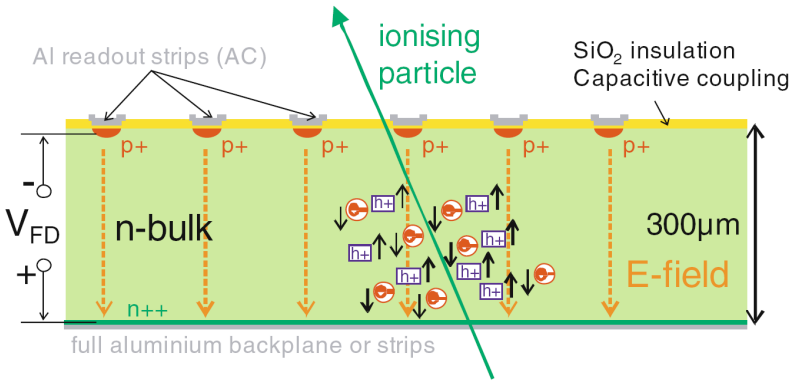

Figure 2.5: Principle of operation of a silicon detector. A particle crossing the device produces along its trajectory electron-hole pairs that are collected by the metal electrodes and constitute the electrical signal. The read out elements are divided in strips in order to obtain a position-sensitive device. Picture taken from [11].

coming from the p side of the junction. Therefore this process creates a region across the junction where fixed charged ions are located. This region is depleted of free charge carriers and is called the *depletion region*. The migration of free carriers creates at the same time a potential barrier that contrasts the diffusion process until a state of equilibrium is reached. The voltage corresponding to the potential barrier is known as built-in voltage V_{bi}.

An external voltage, V_{bias}, of the same sign of the built-in one can be applied to the junction to increase the width of the depletion zone. In this situation the junction is said to be reverse-biased. If a sufficiently high voltage is applied, the whole silicon volume is depleted of free carriers. The voltage at which this condition is reached is called the *full depletion voltage*.

Under equilibrium conditions, electron-hole pairs are thermally generated everywhere within the volume of the crystal and are separated by the electric field created by the bias voltage giving rise to the *leakage* or *reverse current*.

A reverse-biased p-n junction can be used for particle detection. As a particle traverses the depletion zone, the charge released can be collected under the effect of the electric field and read out. On the other hand, the charge created in the non depleted zone recombines with the free carriers and is lost. Hence in silicon detectors the crystal volume is usually fully depleted.

A simplified view of a silicon detector is depicted in Fig. 2.5. An n-type silicon bulk serves as the active volume for the detector and a

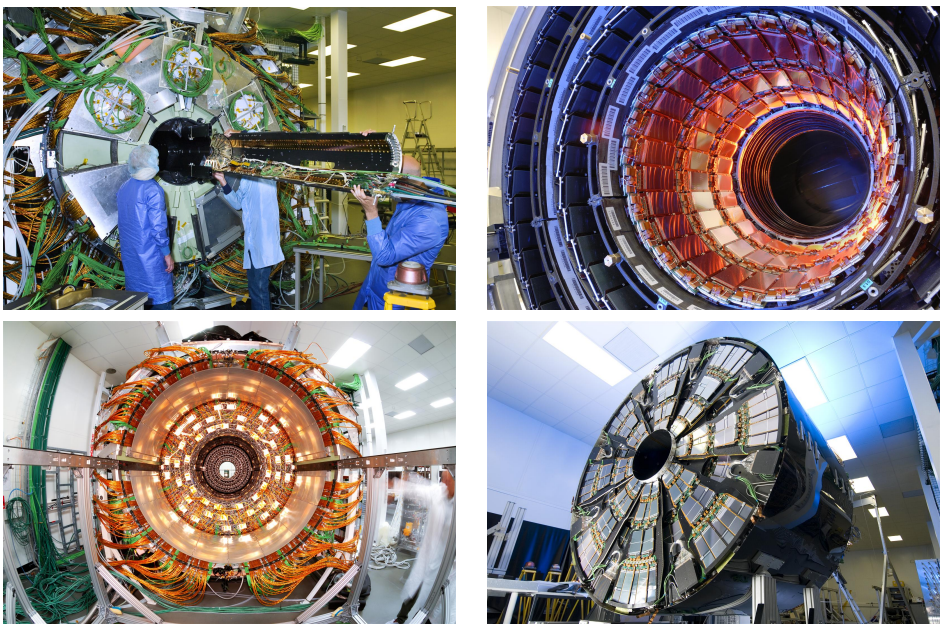

Figure 2.6: Test of the insertion of the pixel detector inside the Strip Tracker, and views of the Tracker Inner Barrel, Tracker Outer Barrel and Tracker End Cap at the end of the integration process.

heavily doped p^+ implantation is used to deplete the junction of free carriers. The junction is completely depleted by applying a reverse bias voltage V_{bias}.

A charged particle crossing the detector creates along its trajectory electron-hole pairs that drift towards the metal electrodes. This charge migration induces a current pulse on the read out electrodes and constitutes the basic electrical signal. The total integral of the current is proportional to the number of produced pairs and hence to the energy loss of the particle.

In order to obtain position-sensitive devices, the p^+ side is divided into smaller independent elements, constituting an array of narrow strips, each one equipped with an independent read out circuit.

2.2.2 Structure and Geometry

The SST is divided into four sub-partitions: two barrel structures and two end cap pairs. The two innermost sectors are the Tracker Inner Barrel (TIB), the cylindrical section, and the Tracker Inner Disks (TID), the end cap. The two outer detectors are the Tracker Outer Barrel (TOB), the cylindrical section, and the Tracker End Cap (TEC) (see Fig. 2.6 and 2.7).

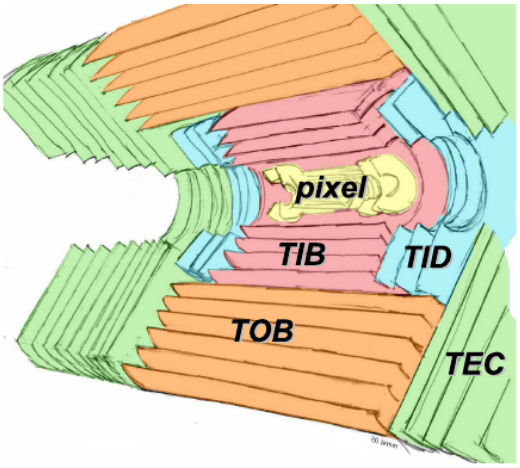

Figure 2.7: Pictorial view of the CMS tracker. The different colors identify each sub-detector: TIB (pink), TID (cyan), TOB (orange) and TEC (green). The pixel detector is yellow.

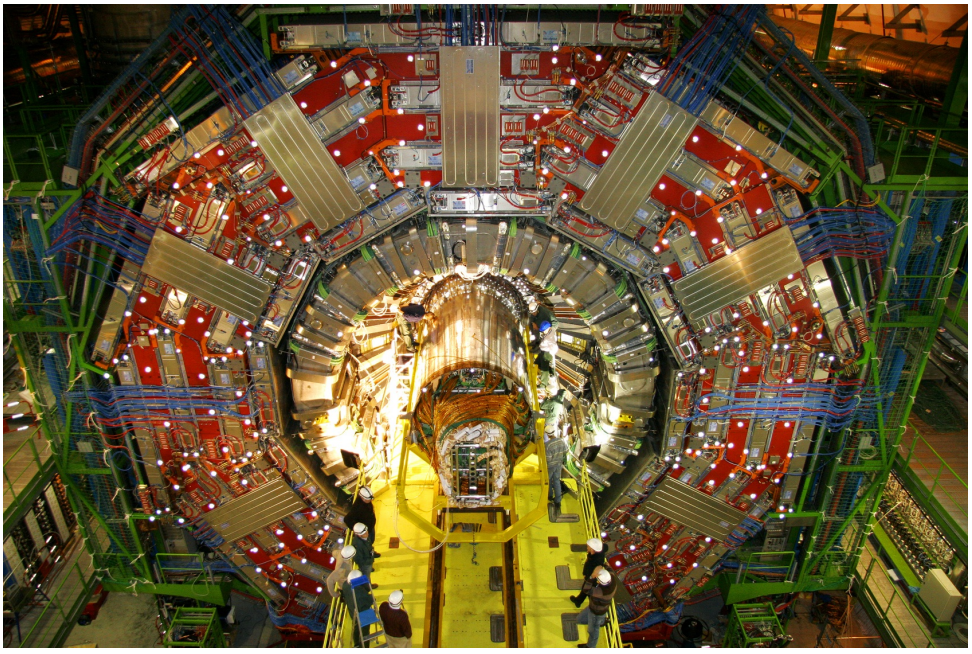

Figure 2.8: Insertion of the tracker inside the CMS detector.

The four regions were developed and integrated by three different communities with a certain degree of independence: the TIB/TID, the TOB and the TEC communities. While sharing some basic elements, the three collaborations implemented some specific choices.

The SST basic sensitive unit is the module, which comprises a silicon microstrip sensor and the associated electronics. Modules are arranged in layers (in the barrel) and in disks (in the end caps). Since some part of the surface of each module is reserved for the electronics and input/output connections, the geometrical position of the modules is designed to cover each layer with the sensitive area of at least one module without leaving any blind region. The SST is composed of 15 232 modules covering an overall surface of 206 m^2. In terms of silicon surface, the CMS tracker is the largest silicon tracker ever built.

Tracker modules can be either single-sided or double-sided. Double-sided modules are made of two independent single-sided modules glued together back-to-back with a relative rotation of 100 mrad respect to each other. Single-sided modules allow only the measurement of (r, φ) in the barrel and (z, φ) in the end cap. Double-sided modules can measure the three coordinates.

The modules are integrated in larger structures (*sub-assemblies*) which implement mechanical support, cooling and connection with the control electronics and power supplies. Regarding communication with the DAQ control system, a branch of modules sharing the same control node, termed Communications and Control Unit (CCU), form a control group.

2.2.3 Front End and Readout Electronics

The main DAQ device is a custom designed circuit called Analog Pipeline Voltage-mode chip (APV25), which amplifies and samples the microstrip signals. When the APV25 receives the trigger signal, it transmits its analog output over an optical fiber to the readout components outside the detector which performs the digitization.

APVs sample the signal from microstrip sensors synchronously with the clock and, when triggered, serialize these signals on their analog output lines. The APV25 has 128 analog inputs, each one connected to a different microstrip of the sensor. The signal is sampled at the full LHC frequency of 40 MHz and the sampled pulses are placed into the analog pipeline consisting of 192 cells. When the pipelines are filled, data is overwritten by the new samples.

Upon receiving a trigger, all the 128 signals are multiplexed into a single analog output. The signals from each pair of APVs are then multiplexed again by the APV multiplexer into a single 40 MHz analog line.

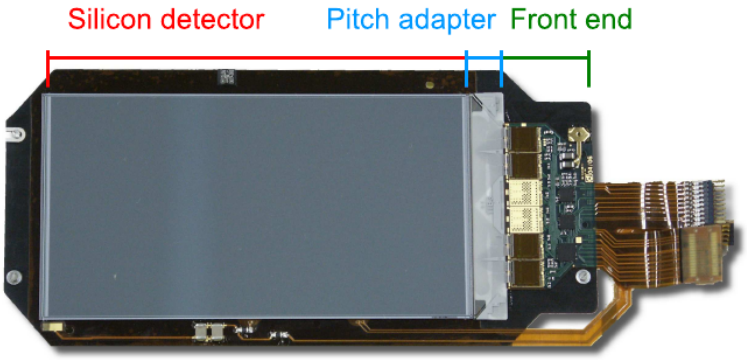

Figure 2.9: Picture of a single-sided TIB module. Each module has 512 or 768 strips that are read out by a custom-designed circuit that, when triggered, transmits analog data over an optical fiber.

The electric to optical conversion is done on a dedicated module, called Analog Opto-Hybrid (AOH), directly connected to each detector module.

The optical fiber outputs are joined together by special extensions (called *ribbons*) of 12 fibers that are brought to a panel outside the tracker (*patch panel*) where the fibers are further packed in *multiribbon* cables. These cables bring the analog signals to the Front End Drivers (FED), housed in the USC, which perform the signal digitization and the first signal processing.

2.2.4 Radiation Damage Effects

The CMS tracker silicon detectors will operate in a high radiation environment caused both by particles produced in the primary proton-proton interaction and by neutrons emitted from the calorimeters surrounding the silicon tracker itself. Because of the high collider luminosity, the foreseen radiation level is much higher than in previous hadron colliders.

The macroscopic effects of radiation damage to a silicon detector are an increase of the leakage current, the change of the full depletion voltage, increased noise and reduced charge collection efficiency. The bias voltage might be adapted after several years of operation of the silicon strip tracker, but it should never exceed 500 V for all the modules, including the inner ones that are exposed to higher fluence.

To reduce the effects of radiation damage, the silicon detectors will be kept cold, working at a temperature of $-10\,^{\circ}\mathrm{C}$. Only during limited maintenance periods, the detectors will be "warmed" up to above $0\,^{\circ}\mathrm{C}$. The electronics dissipate power, so a cooling system is needed to maintain

this temperature. A coolant fluid at a temperature of $-25\,^{\circ}\mathrm{C}$ reaches all the devices of the tracker. A thermal shield (called thermal screen) separates the tracker volume from outer detectors. In order to safely operate the detector under these conditions, temperature and humidity should be constantly monitored.

2.3 The Front End Layer for the Control of the CMS Tracker

2.3.1 Power Supply System

The Power Supply System for the CMS Silicon Strip Tracker [12, 13] provides High Voltage (HV) bias and Low Voltage (LV) power to the silicon strip modules of the detector. The HV is needed to bias the modules and allow particle detection. The LV is needed to power the APVs.

The power supply system must be able to deliver ~ 15 kA inside the tracker volume, for a total power of ~ 33 kW. The design of the power supply system had to satisfy numerous constraints. The choice of the electronics was governed by requirements on noise and on the response to sudden current transients. The need to configure the powering of the system with a fine granularity and to setup small scale integration centers for testing different parts of the tracker called for a modular and distributed design. Under this scheme each group of homogenous modules are powered by an independent unit, performing low level safety tasks.

The power supply modules are installed in racks located on balconies at both sides of the detector, at around 10 m distance from the interaction region, in the UXC. The hostile environment in the cavern implies that the power supplies are inaccessible during the operational period of the experiment.

From the control point of view, these constraints impose the need for a reliable control and diagnosis system. In addition to providing the remote control, the DCS should permit the identification of problems and solve them remotely, when physical access to the experimental cavern is not possible.

Even though the power system is designed to take autonomous safety actions in case of critical conditions, current and voltage thresholds are configurable over a large range. Proper configuration of the channel parameters, which is one of the main control system duties, is therefore absolutely critical for the detector safety.

The power supply system was implemented by CAEN[1], a leading company in the field of sophisticated electronic equipment for Nuclear Physics research. The power supply modules are completely custom made for the CMS tracker and are integrated within the new EASY (Embedded Assembly SYstem) scheme. Under this scheme, power supplies are directly located in the hostile area and remotely controlled by a supervisor mainframe placed far away in standard environment.

Each silicon strip tracker module requires one HV regulator to bias the sensors and two LV regulators (one at +2.5 V and one at +1.25 V, respectively for digital and analog electronics) to power the front end chips and the other circuitry. Modules are grouped into 1944 "power groups" consisting of 2 to 12 modules (12 to 56 APV). The grouping criteria are governed by the mechanics and by the density of channels. A finer granularity is adopted at low radii, close to the interaction point. The digital control optoelectronics requires a distinct powering system. Control services are grouped in 356 CCU-rings, each one requiring a 2.5 V power source.

The power supply system building block is the Power Supply Unit (PSU) providing the two low voltage sources and two high voltage supplies to one power group. To increase flexibility, the PSU has two independent high voltage channels, each one powering about half of the modules in the power group. Two PSUs are combined into one Power Supply Module (PSM) of type A4601H. Another PSM model (A4602), providing 4 low voltage (+2.5 V) channels is used to power the control rings and can be included in the same powering scheme.

Up to 9 PSMs are lodged inside one standard crate (19″ wide, 6U high). Up to 6 crates (grouped in one rack) can be controlled by one Branch Controller through a CAENBUS communication link (a custom implementation of the CANBUS). Up to 16 branch controllers can be inserted into a CAEN mainframe (SY1527) that can communicate via ethernet to the supervisory layer of the control system (see Fig. 2.10). SY1527 mainframes run a standard Linux operating system that acts as a server to the supervisory layer. A specific OPC server is dedicated to communicate with the mainframe.

The Power Supply Units are served by two independent +48V DC sources, one for the service electronics (called service power) and one for the final power stage of the regulators. In the base implementation chosen by the tracker, each 48V power source serves a crate of power supplies. Another 48V source provides service power to each rack of crates. A CAEN 48V converter board[2] hosts two independent 48V channels that

[1] Acronym for the Italian *Costruzioni Apparecchiature Elettroniche Nucleari*.
[2] CAEN model A3486, also known as "MAO"

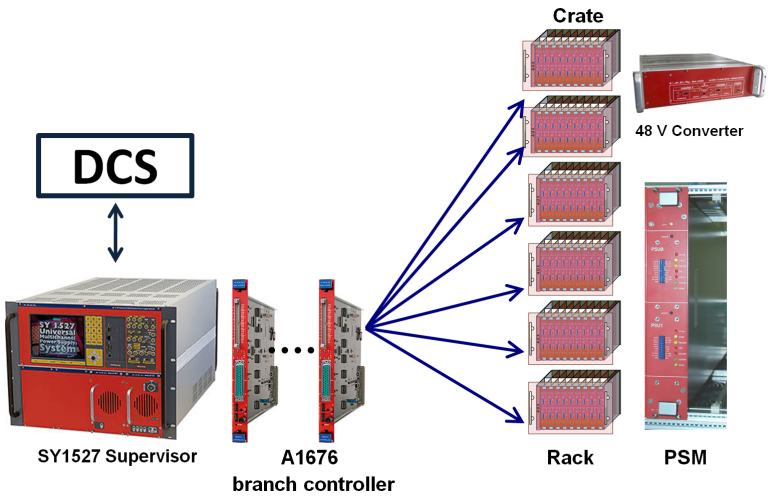

Figure 2.10: Control chain for the power supply system. The DCS is connected to a supervisor mainframe. The branch controllers provide connections to the racks that are placed in the hostile environment. Each rack is composed of several crates that host the Power Supply Modules, providing LV and HV to the detector. The 48V converters, providing the service power to the Power Supply Modules, are located in the same crates.

can be used either as a service or as a power source.

Two types of cables are used to route the power: the power groups are connected via a Low Impedance Cable (LIC) while the control power goes through a Power Long Control Cable (PLCC). Power cables cover a total distance of about 50 m from the PSUs down to the tracker. The same cables route the LV power, the HV, the sense wire and some environmental information (temperature and humidity) from the tracker structure. The connection is made by one 40-45 m long low-impedance cable placed mostly outside the CMS volume, and one $\sim$ 5 m long one lying entirely inside the tracker volume. The two parts are connected at the periphery of the detector in Patch Panel 1 (PP1) connector boards.

Given the considerable length of the power supply cables, a significant voltage drop is expected for the low voltages, which have higher currents. To be able to compensate the voltage drop, a sense wire reports the measured voltage on the load to the regulator that can adjust the supplied voltage at the connector side in order to ensure the nominal voltage. Voltage drops up to 4 V can be compensated by the regulators over up to 150 m long cables.

The PSUs are controlled by an 8-bit microprocessor. For each of the four channels it is possible to set the voltage. For the LV source, this can be adjusted to within $\pm 5\%$ of the nominal value, while for the HV

Figure 2.11: The inner structure of a A4601H Power Supply Module. Each module hosts two boards. Each board has 2 LV and 2 HV channels serving one power group of the tracker.

source it can be set to a value between 0 and 600 V. Other configuration parameters are the limit for the current and for the sensed voltage and the reaction time for the channel trip in case of over currents. The expected current is proportional to the number of APVs served by the power group, hence the current limit should be configured differently for each Power Supply Unit. In the HV channels it is also possible to set the ramp up and ramp down speed. The parameters that can be monitored are: the status of a channel, given as a bitfield of conditions, the sensed voltages and currents, and the voltage at the connector for the low voltage channels.

The Power Supply Unit implements some internal safety measures that switch off the channel in case of dangerous situations and set its status to an error condition. Special attention is devoted to prevent over-voltages affecting the powered electronics whenever an abrupt current consumption variation occurs. This is a common problem experienced when electronic devices are operated from a long distance using the sensing technique. The PSUs ensure voltage recovery within 0.5 ms.

In case of a critical condition (over current, over voltage or under voltage), the channel is switched off automatically (trip) and an error bit is set in the status register. Since the requested reaction times are less than one millisecond, the mainframe cannot implement the safety features because the delay would not be acceptable. Hence, the safety procedures are implemented in the PSU microprocessor.

When the internal safety acts on the LV channels, the HV channels are also switched off. The two LV channels can only be switched on or

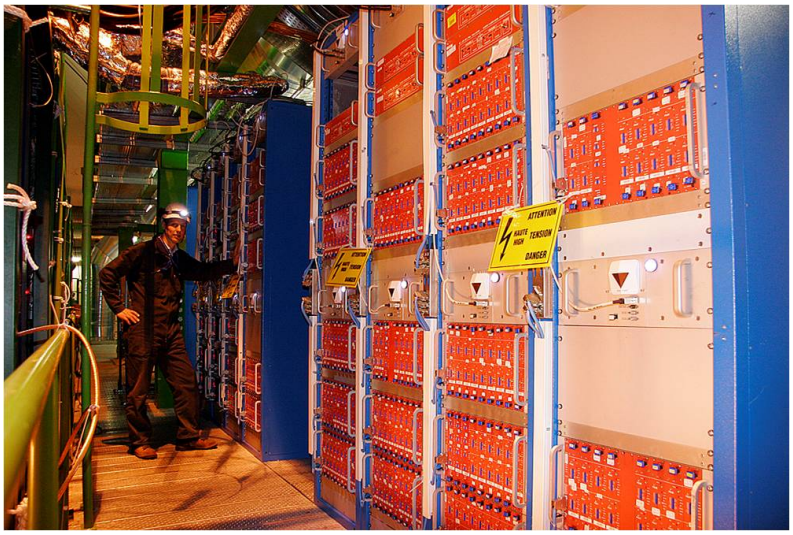

Figure 2.12: The racks with the power supplies modules installed in the balcony of the UXC.

off at the same time.

Each crate provides 5 interlock lines that must be kept at the nominal voltage of 24 V by the safety system to enable operation of the Power Supply Modules. When the signal in the interlock line is removed, all the Power Supply Units ramp down all the voltages to zero. According to the "positive safety" approach, the normal condition is signaled by a "high" level, while the alert condition gives "low" levels. In case of hardware failure, e.g., broken interlock cable, the power supply system is taken to a safe state.

One interlock line is used for the "crate interlock" which causes a fast interlock at the speed of $200\,V/s$, while the standard four lines cause a normal interlock at the speed of $50\,V/s$. The Power Supply Module can be configured to respond only to some of the four interlock lines. This configuration is designed to allow a partial interlock of the crate. However, this feature is not used in the Tracker Safety System because the safety might be compromised in case a PSM is replaced with another one with different configuration.

The final power supply system of the CMS Silicon Strip Tracker makes use of 986 boards of type A4601H, 110 boards of type A4602 and 79 power converters (A3486). These are spread over 139 crates and 29 racks (placed in 6 different balconies), which are finally supervised by 4 mainframes (SY1527). For performance reasons, the mainframes are not fully loaded. Figure 2.12 shows the setup in one balcony and figure 2.13 shows the LIC and PLCC cables connected to the backboards of the power supply

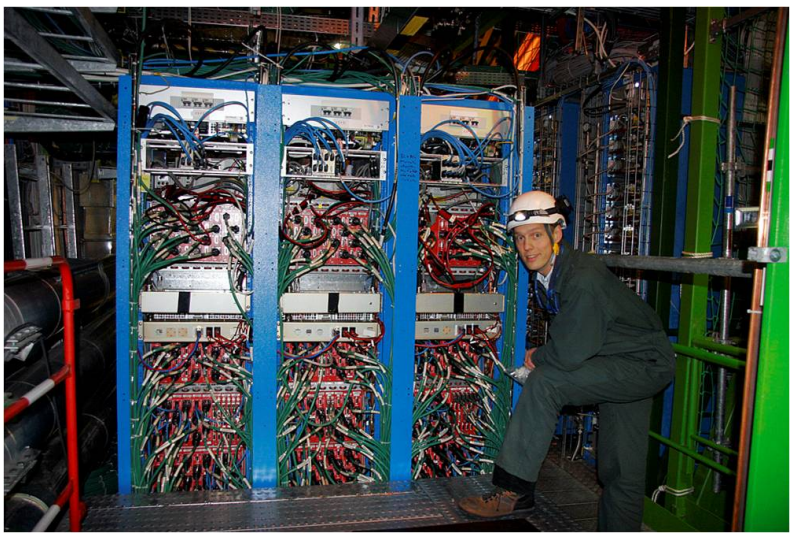

Figure 2.13: The cabling of three power supply racks. The power cables are routed from the power supply modules to the PP1 connection boards.

modules.

2.3.2 Cooling System

The cooling system is designed to remove the heat generated by the electronics and to keep the tracker at the nominal temperature. For cooling purposes, the strip tracker is divided into 180 cooling loops, served by two detector cooling units located in the UXC. The Heat Transfer Fluid circulating in the cooling lines is a fluorocarbon (C_6F_{14}). After circulating into the detector, the fluid returns into the cooling unit to be chilled again. The working temperature can be set in the chiller. Sensors for temperature and pressure, flowmeters, and tank level measurements are integrated in the chiller and report their values to a PLC system. These parameters are monitored by the cooling plant internal safety system that is able to switch off the cooling plant in case of critical conditions. The cooling units are connected to the Tracker Safety System and interlock the entire tracker if one unit is not working.

Several parameters can be read and set in the cooling plant PLC, including the state of the valves and the setpoint for the working temperature. The DCS is responsible for monitoring the PLC values and should provide the cooling experts with the means of configuring the cooling plant.

2.3.3 Tracker Safety System

The safety of the tracker has to be ensured by avoiding all situations that may cause permanent damage to the detector hardware or affect its lifetime or performance [14]. Clearly safety must take precedence over data taking and can take actions that stop the tracker operation in case of critical conditions. On the other hand, all effort has to be taken to prevent the critical conditions from happening, in order to avoid losing precious time and data during physics runs.

The main environmental measurement that has to be monitored is the operating temperature. Temperatures outside the specified operating range can occur in case of problems with the cooling plants or in individual cooling loops (e.g., blockage). Over temperatures, in particular after the tracker has been exposed to significant irradiation, can result in reduced charge collection efficiency and eventually preclude tracker operation[3]. Low temperatures can result in permanent mechanical damage, or, in the presence of sufficiently high humidity, can lead to condensation.

The humidity is a critical parameter. Since the tracker is operated at temperatures well below 0 °C, it must be kept most of the time in a dry air or nitrogen atmosphere but it will unavoidably be exposed to standard atmosphere during any access. To prevent water accumulation inside the mechanical structure or on the modules, the dew point must stay well below the operating temperature. The relative humidity is hence constantly monitored.

External global conditions must also be taken into account in the safety system. Typical examples are the cooling plant and dry air system status and the signals coming from the general CMS safety system, dealing with events such as fire or flooding of the cavern. Communication with external systems is unidirectional and limited to simple boolean (OK/NOT OK) values.

In case of a critical condition, the safety system interlocks the power supplies. Hardware action on the cooling system is only foreseen in case of failure of the dry gas system. While the power supply interlock has a fine granularity, the interlock of the cooling system affects all detector cooling units.

The TSS (Tracker Safety System) is implemented in a self-contained independent Siemens PLC-based system, operating on the information provided by over 1000 hardwired temperature and humidity sensors. In case of critical environmental conditions, the system acts by interlocking

[3]Silicon sensors are operated as reverse-biased p-n junctions, where the whole sensor volume is depleted of free charge carriers. Particle radiation changes the resistivity of the bulk material, thus the voltage needed to fully deplete the sensor increases.

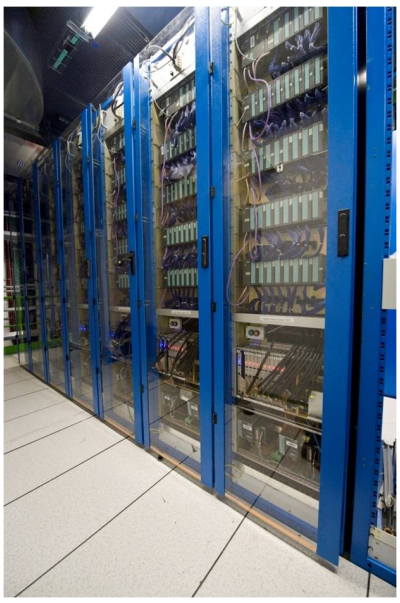

Figure 2.14: The PLC racks for the Tracker Safety System, installed in the USC.

the power supplies via a hardwired connection from PLC relays to the crates, cutting the 24 V alive signal by opening the magnetic relay. Three main types of environmental sensors are used: PT1000 thermometers, thermistors (of two different models, Murata and Fenwal) and specialized relative humidity sensors (HMX), chosen for their radiation-hardness properties.

Due to its size and complexity, the safety system is split into six independent systems, each one serving one side ($z+$ or $z-$) of the three subdetectors: TIB/TID, TOB and TEC. A separate PLC system (Monitor PLC) is used to read additional environmental information from around the whole tracker. Finally, a master PLC system deals with 'global' critical conditions such as cooling plant failures, main power failure and PLC internal failures.

Upper and lower limits can be configured and disabled individually for each sensor. The state of the sensor (ok/out-of-limit) is defined depending on these limits.

The interlock logic is designed to allow partial regional shutdown. In this way track reconstruction is still possible in case of a local problem, as long as the affected region is not too large. Interlock granularity is configurable in the PLC by means of interlock groups. Each group is composed of a map between sensors and relays and a majority threshold. Each group defines a rule: if the number of out-of-limit sensors in the group reaches the majority threshold, then the corresponding relays are

opened and cause the interlock of the connected crates. The sensors and relays of each group are specified with bitmasks.

The supervisory layer is responsible for downloading the configuration of the interlock scheme (sensor and relay bitmasks and majority thresholds for each group) to the PLC. This is a safety-critical operation that must be absolutely protected from misconfiguration by non-experts and accurately checked after each reconfiguration. A rigorous access control ensures that the configuration process is available only to experts. An accurate checking procedure has been developed to guarantee the correctness of the interlock configuration (see Sec. 4.8).

The PLC code is designed to meet the two contrasting aims of stability and flexibility. Nothing should cause a delay or a failure of the interlock. However, the scheme should still be configurable, adapting to variations of the constraints. Thanks to its configurability, the same general PLC code can be used for the six independent PLC systems, each one reflecting the specific cabling.

In order to have a flexible system, bidirectional communication with the supervisory layer is unavoidable. Nevertheless, all I/O communication is performed in isolated separate memory spaces, avoiding interference with the algorithms to determine whether an action must be taken. As an additional precaution, the master system, completely isolated from supervisory layer, checks the heartbeats of all the other PLC racks. In case of malfunctioning of one PLC rack, the master system is able to interlock the relevant region of the tracker, bypassing the affected PLC rack.

The PLC continuously executes the same sequence of instructions (called a PLC cycle). Each PLC cycle takes care of copying the input data from the sensor to the memory bank where it can be read by the supervisory system. It then analyzes the data coming from the sensor generating a list of flags and receives the acknowledgements. The flags can activate external signals (executing the commands). A maximum time for the execution of a cycle is introduced for safety reasons.

Design of the Interlock Cabling Scheme

Temperature sensors in the tracker can be classified into three types, depending on their position in the tracker structure. In TIB and TOB, the temperature sensors are located close to the cooling lines ("on liquid"), while in TEC they are located on the modules and directly measure the silicon temperature ("on silicon"). For this reason, when the tracker is powered and the electronics is configured, the sensors on the TEC consistently measure a temperature that is $\sim 7\,^{\circ}\mathrm{C}$ higher than on TIB

and TOB.

In addition, some sensors in TOB and TEC measure the air temperature, and, with the humidity sensors, can be used to compute the dew points.

The interlock scheme only uses the temperatures on silicon or on liquid. In the case of the TEC, the modules of one control group are served by several cooling loops. The power supply modules of one control group in the TEC must thus be interlocked in case of problems in any of the four cooling loops related to its sector.

The probes corresponding to the same cooling loop (for TIB/TID and TOB) or sector (for TEC) are grouped in the same interlock group and are used to interlock the crate(s) of the corresponding power supplies. Each relay is connected to one *crate block*, that is the set of one to three crates containing the power supplies related to a certain cooling loop or sector.

The hardware granularity is not sufficiently fine for switching off only the power supplies related to a single cooling loop. As a result, the entire crate block, including power supplies powering modules served by other cooling loops, is affected by the interlock when one relay is fired. The supervisory layer implements a protection mechanism (see Sec. 4.5) that switches off the cooling loops or sectors with finer granularity, in order to avoid an intervention of the safety system that would switch off as a side-effect other parts of the detector.

2.3.4 Detector Control Units

Due to high levels of LHC radiation, the major damaging effects on the silicon micro-strip detectors are leakage current increase and change in the detector depletion voltage. For this reason, careful monitoring of the detector environmental conditions is needed with the highest possible granularity, that is, for each individual detector module.

An Application Specific Integrated Circuit (ASIC), called Detector Control Unit (DCU) has been developed for this purpose [15]. Every module is equipped with one front end DCU that measures the detector leakage current, the APV power supply voltages and the temperature on the module at three different points (one directly on the silicon sensor, one on the front end hybrid and one on the DCU itself). A similar circuit on the CCU provides measurements with the granularity of the control group.

2.4 External Systems

Several external systems must be monitored to ensure safe operation of the tracker. Usually the information from these systems is acquired via external projects that are not developed by the CMS tracker control system team.

Although all the critical conditions are handled in hardware, some less severe problems may trigger a software switch-off and inhibit switching on the tracker.

Many of the CMS sub-detectors can operate safely only when beam conditions are good. For monitoring the quality of the beam, several Beam & Radiation Monitoring (BRM) sub-detectors have been designed. These ensure the safety of CMS by causing a shut-down of delicate systems before beam losses can inflict serious damage [16]. The status of the BRM can cause the switch off of the high voltages in the tracker and prevent them from being switched on again.

The status of the LHC accelerator has to be taken into account in the control of the tracker and can inhibit switching on the tracker when the beam is not stable.

The Thermal Screen is the active thermal interface that insulates thermally the tracker from the surrounding detector, the ECAL. This allows the two detectors to be operated at different temperatures (the ECAL runs at +17 °C versus the -20 °C of the tracker). It is controlled by a dedicated PLC that regulates its behavior by means of a sophisticated PID (Proportional Integral Derivative) controller. However, from the point of view of the DCS, the status of the thermal screen can be regarded as binary information (working/not working) that can prevent the tracker from being switched on.

2.5 Data Volume and Expected Change Rate in the Tracker Control System

Control systems for LHC experiments are much more complex than the analogous systems during the LEP era mainly because of size, complexity and data volume. In practice, the data handled in a single sub-detector of an LHC experiment is larger than the data handled in the control system of an entire LEP experiment. The CMS Silicon Strip Tracker is the largest silicon tracker ever built, as well as one of the most complex among all the CMS sub-detectors. This complexity is also reflected in the number of parameters handled in the control system.

The main sources of data for the Silicon Strip Tracker, are the power

supply system, the monitoring of the PLCs in the safety system, the DCUs and the cooling plant control.

The data volume can be measured using different data units. In PVSS, data is organized in tree-structured data points (see Sec. 1.10). Typically, a data point corresponds to a physical hardware device. The properties describing a device correspond to the data point elements. However, some elements can be used to store information for internal processing and not for communicating with the hardware so it is also relevant to count the number of elements associated to a hardware address.

For the CAEN system, 4 PCs are used (see Sec. 4.4) each one handling 2 500 - 3 000 data points for a total of about 10 000 (Table 2.1). Each data point contains several elements, some being connected to an OPC address. The elements connected to a hardware address are about 180 000 (Table 2.2). In the design of the cabling scheme, a large effort has been made to balance the load among the four mainframes and PCs. Considering the number of constraints that had to be taken into account in the cabling, the balancing is more than satisfactory.

Type	**Sys. 1**	**Sys. 2**	**Sys. 3**	**Sys. 4**	**Total**
Channel	1 992	2 244	1 936	2 306	8 478
Board	508	571	493	587	2 159
Crate	32	37	35	35	139
Branch controller	7	8	7	7	29
Mainframe	1	1	1	1	4
Total	2 540	2 861	2 472	2 936	10 809

Table 2.1: Number of CAEN data points. Note the good balancing among the four systems.

CAEN System	**# OPC Items**
1	42 223
2	47 432
3	40 513
4	48 229
Total	178 397

Table 2.2: Number of OPC Items for the CAEN systems. Spread over four PCs, the total number is close to 180 000.

The various types of values read from the safety system total 4 362 data points and about 80 000 addresses (Table 2.3).

Type	# Data Points	# Addresses
Sensors	954	11 334
HMX Sensors	246	1 722
PLC Configuration (in / out)	2 400	18 200
Relays	544	3 264
Groups	138	44 712
Digital Inputs	80	640
Total	4 362	79 872

Table 2.3: Number of PLC data points and data point elements with addresses. Large bitmasks are addressed bit by bit resulting in a large number of addresses in the data points representing groups.

The DCUs are about 17 000, with 15 000 front end DCUs corresponding to the number of modules. Each type of DCU has a variable number of elements leading to a total of about 100 000 elements (Table 2.4).

Type	# Data Points	# DP Elements
Front end DCU	15 148	90 888
DCU on CCU TOB	1 582	6 328
DCU on CCU TEC	716	2 864
Total	17 446	100 080

Table 2.4: Number of DCU data points

However, the performance depends not only on the number of addresses but also on the change rate. In the communication layer, devices are typically read out with constant refresh times. Communication at this level is rarely event driven, because the limited resources at the front end layer limit the possibility to implement an event driven protocol. The front end devices have an intrinsic readout delay that must be taken into account. For example, the mainframe takes at least 500 ms to update all the power supply parameters of the boards (see Sec. 4.9.1 for details). The communication with the front end hardware uses a LAN, which causes a further delay. Therefore, the refresh times in SCADA systems are typically of the order of seconds. Every action requiring a shorter reaction time should be implemented in hardware at the front

end layer. In large systems, the refresh rate should be set on the basis of the parameter criticalness, but it is in general useless to set refresh times under 1-2 seconds.

The addresses can be divided into input and output addresses. The inputs correspond to the monitored data read from the controlled devices and the outputs are the commands that the control system sends to the devices. The commands do not have an impact on the communication performance, since they are in general much more infrequent than the typical updating time of the input parameters (although commands tend to come in bunches).

For the settings, the same address is replicated twice, as an input and as an output. In this way it is possible to set the element and have the feedback that the write operation was successful. This choice implies that the input parameters are by construction more numerous than the output parameters.

The input elements can be roughly divided into three categories. Very static elements, such as the serial numbers of the boards and the types for the PLC probes, change very rarely. The readings of the user settings can change in case of reconfiguration of the hardware but are not expected to change during standard operation. Finally, some values describe the present state of the hardware and must be monitored during operation with a short refresh time. Typically these three classes of elements can be read with different polling times. This allows for a good response in the fast-changing parameters at the cost of having to wait longer for feedback at the moment of configuration. Table 2.5 shows an example of the distribution of OPC items in the three polling groups (only the input items are taken into account), showing how the number of parameters read with a fast refresh can be reduced to less than a third. In the case of the PLC readings, the elements can also be partitioned into fast and slow polling groups. The items that need to be in the fast group are the online values of sensors and relays giving a total of 1 744 elements (Table 2.6). For further discussion about the optimization of communication with the hardware, see Section 4.9.

	Sys. 1	Sys. 2	Sys. 3	Sys. 4	Total
Fast ($\sim$ 2 s)	7 870	8 694	7 566	9 090	33 220
Slow ($\sim$ 6 s)	16 364	17 410	15 440	18 354	67 568
Ultraslow ($\sim$ 60 s)	2 948	3 336	2 880	3 412	12 576
Total	27 182	29 440	25 886	30 856	113 364

Table 2.5: Number of input OPC Items grouped by polling times

Polling speed	# Addresses
Fast - Sensors ($\sim$ **2 s**)	1 200
Medium - Relays ($\sim$ **5 s**)	544
Slow ($\sim$ **20 s**)	$\sim$ 40 000

Table 2.6: Number of input items in the PLC grouped by polling times

Chapter 3

The CMS/LHC Detector Control System

This chapter describes the architecture of LHC Control Systems and how the general SCADA architecture presented in Chapter 1 is adapted to a HEP environment.

The large amount of data to be handled and the need to introduce HEP-specific tasks call for a scalable and extensible SCADA solution.

The overall control system of an experiment is built by integrating many independently-developed projects. To ease the integration process and avoid the duplication of work, a framework providing common functionalities and guidelines has been developed at CERN.

The complex structure of the experiment requires various sub-systems to be operated, debugged and commissioned in parallel. For this reason, the control system is organized in a hierarchical structure with a flexible partitioning mechanism.

3.1 Scope of the Detector Control System

3.1.1 Architecture of the Online System (DAQ, DCS, Run Control)

The operation of a HEP experiment makes use of two main software systems: Data Acquisition (DAQ) and Detector Control System (DCS). Data Acquisition is the final objective of the experiment. The DAQ is responsible for acquiring and processing data from the front end electronics and storing it in a proper format for offline analysis. The DCS is responsible for monitoring and controlling the auxiliary hardware, not directly related to data acquisition. An additional interactive software, the Run Control and Monitoring System (RCMS) [17], controls and monitors the

collection of data from the detector, ensuring the proper configuration of the DAQ processes. RCMS is also responsible for coordinating DAQ and DCS.

3.1.2 Detector Safety System

Since the DCS is a complex software subject to failures (e.g., LAN problems, PC failures) and prone to programming mistakes, the detector safety cannot rely on the DCS and must be guaranteed by an independent Detector Safety System (DSS).

The DSS has the primary goal of detecting any abnormal and potentially harmful situation and take protective actions in order to minimize the consequent damage to the experimental equipment [18]. It should be highly reliable and available, simple and robust. It must operate permanently and independently of the state of the DCS.

The DSS is implemented in a self-contained hardware system that is able to react to critical conditions within a bounded time, typically by switching off parts of the detector. It is based on PLCs that operate on the information coming from hardwired sensors. Networked information is not considered reliable and is not used in the DSS.

The DSS is not the only system responsible for safety-related issues. In the case of CMS, two additional safety systems (CERN Safety System and CMS Safety System) focus on people's safety and on addressing widespread severe problems such as fire or flooding of the cavern. These two additional safety levels are independent from the DSS and act on general services, such as the general power to the entire experimental cavern.

Since the detector safety is managed by an independent hardware system, the experiment can safely run without continuous human supervision. The human interface to the DCS, though important to monitor and control the status of the detector, is not critical for its safety. No critical situation should require human intervention to bring the system to a safe condition.

3.1.3 Role of the Detector Control System

The DCS is the software that enables coherent and safe operation of a HEP detector. The DCS is responsible for providing continuous supervision of the detector's auxiliary hardware. It is used to bring the detector to the "ready for physics" state, an acceptable condition for data taking. The DCS must be able to display the status of the detector at various levels of detail, allowing different interactions depending on user exper-

tise. The main users of the DCS are the experiment operators ("DCS shifters") and the sub-detector (e.g., calorimeter, tracker, muon system) or sub-system (e.g., power supply system, cooling system) experts.

The DCS provides the main interface for the experiment shifter to monitor and control the detector. This interface must be able to summarize effectively the status of the system, conveying the information from hundreds of thousands of monitored parameters in an overall state, understandable by a non-expert operator. The DCS must be able to respond to high level commands and translate them into the proper sequence of low level commands submitted to the controlled hardware in order to guarantee safe and effective operation.

At the same time, the DCS is used by the sub-detector or sub-system experts for analyzing and configuring the state of the system in greater detail.

The DCS is responsible for signaling to the user any anomalous and potentially dangerous values (too high temperatures, unexpected currents, etc). Additionally, even if the safety of the detector is ensured by DSS, the DCS can take some automatic actions in order to bring the detector into a safer state. Protection actions programmed in DCS may avoid the widespread intervention of the safety system, thus minimizing the downtime due to critical conditions. The DCS should also prevent the user from giving commands that would result in an unsafe state, by inhibiting some actions under certain conditions. Finally the DCS is used to properly configure the front end hardware, including the safety parameters.

The DCS must provide the supervision of the experiment 24 hours a day, allowing for the coherent combined operation of the different subsystems. A working and efficient DCS is needed to reach the data-taking phase and to maintain stable and safe running conditions. The DCS software should run continuously for the duration of the experiment (over ten years). It is an essential component for the success of the experiment.

3.2 Control Layers

The software architecture for the control of a HEP experiment is organized in a communications stack where the software at each layer acts as a server to the higher level software and as a client to the lower level one (see Fig. 3.1). The usage of various levels of abstraction eases the implementation of the software at all levels and favors the distribution of the work among different developers. The modularity facilitates the integration of different solutions and gives the freedom to adopt commercial

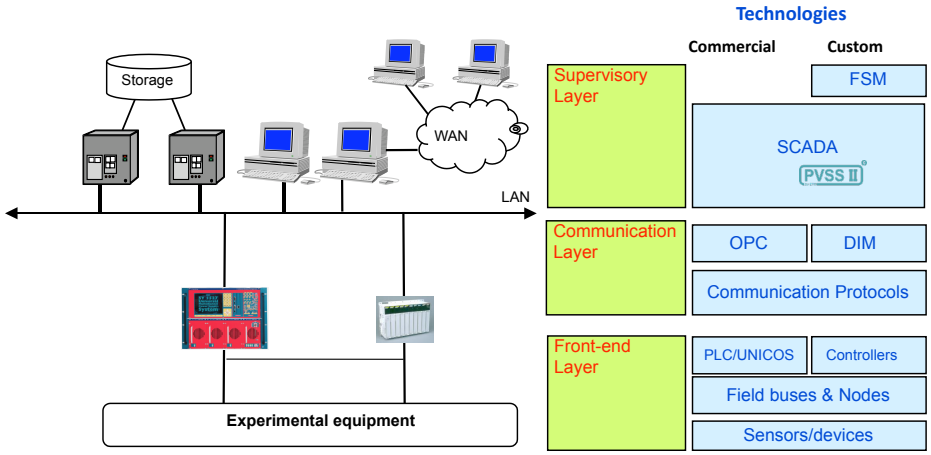

Figure 3.1: Controls Technologies in the LHC era. The software part is organized in a communications stack that can be divided into three main layers (supervisory, communication and front end). The figure summarizes the most common protocols and technologies used at the various levels. *Modified version of a figure by CERN IT/CO - based on an original idea from LHCb.*

software where a suitable application exists and write custom software where needed.

The supervisory layer of the entire CMS experiment uses about 100 PCs (10 for the tracker) installed in the service cavern. The user interfaces run on additional workstations in the control room. Finally, summarized real-time information is published on the web. Historical data is archived in dedicated database servers and can later be accessed from a web interface.

3.3 Criteria for the Selection of the SCADA Product

All LHC control systems are based on the commercial SCADA product PVSS-II (see Sec. 1.10).

A discussion of the criteria adopted in the selection of the SCADA product can be found in [19]. A review of PVSS after four years of operation at CERN is described in [20].

3.3.1 Scalability

Scalability is probably the most fundamental item in the list of LHC control requirements. Some reviewed SCADA products resulted unfit

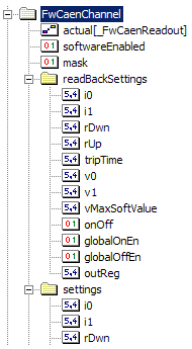

Figure 3.2: Example of tree-structured data point type for the control of a power supply channel. The information is hierarchically organized to reflect the structure of the modeled device.

for the control of large LHC systems because of built-in limits in the number of controllable items. PVSS has no built-in limit and allows for the interconnection of several projects to create a distributed coherent large control system. Extensive tests [21] were performed to show the possibility of connecting more than 100 PVSS systems according to a hierarchical approach where every system is connected to all its descendants. This hierarchical connection is the expected configuration of a large control system where the main supervisor system is connected to all other systems and the sub-system supervisors are connected to all the related PCs.

3.3.2 Structured Runtime Database

PVSS provides a structured runtime database where information is organized in so-called data points. A data point (DP) is a hierarchically structured set of data point elements (DPE) of primitive types. Each data point is an instance of a certain data point type (DPT). The DPT defines the structure of the data point as well as the names and types of the elements (see Fig. 3.2).

The advantage of using data points rather than data organized in a flat namespace is that the former organization favors the reusability of the components and the usage of generic code. In fact, even if the programming language used in PVSS is not object-oriented, the code can be organized in generic functions that take as a parameter a data point name (of an expected type) and execute some action on the specific data point instance. Moreover, PVSS supports generic user interface objects, called *reference panels*, that allow a graphical representation of a device to be developed once and instantiated as many times as needed

at runtime. This feature is very important for the HEP environment, where a large number of homogeneous devices have to be controlled. Still, PVSS does not follow an object-oriented architecture for data handling, because any concept of class, inheritance or polymorphism is missing in the data point types.

PVSS does not distinguish between hard points (corresponding to the communication with the devices) and soft points (corresponding to computed or auxiliary information). Instead, PVSS supports the configuration at the level of data point element of several properties, such as hardware address (via a particular driver), smoothing, archiving, alert handling, and connection to the values of other elements.

3.3.3 Extensibility

Extensibility of the SCADA product is a key requirement in a HEP environment where typical SCADA tasks should be complemented with more HEP-related tasks. The extensibility of PVSS comes with its programming language, called CTRL, that was used to program general HEP-related libraries (see Sec. 3.4). Moreover, C++ PVSS APIs allow the development of custom PVSS managers implementing specific functionalities.

3.3.4 Constant Development

An industrial control system is typically developed in a centralized manner by experienced SCADA experts. Once it is installed and operational, it is rarely modified. In contrast, the DCS is developed in parallel by different teams. It is constantly modified, often in parallel to the operation of the experiment. Changes to one component should be available in the integrated system without stopping the operation of the other components. The distributed system features of PVSS, with automatic update of the changes in the remote namespaces, are suited to these needs.

3.3.5 Ease of Use of the Scripting Language

PVSS control scripts and user interfaces are programmed using a specific C-like interpreted language, named CTRL. This reduces the learning phase of new developers who are already familiar with C and makes it possible to use documentation systems like doc++ intended to be used with C or C++. On the other hand, the CTRL language has some limitations because it follows a pure procedural programming paradigm with no object-oriented extensions nor support for exceptions.

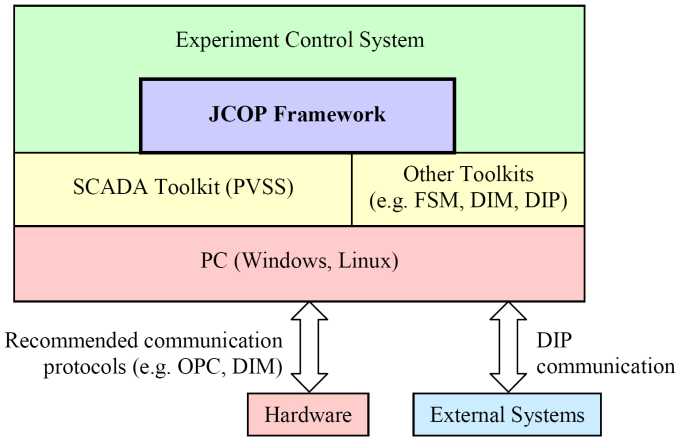

Figure 3.3: Architecture of the JCOP framework. JCOP provides common libraries built on top of PVSS. The control system developer uses the tools from the framework and PVSS to develop his application.

3.3.6 Cross-Platform

While most of the modern SCADA products are only available for Microsoft Windows, PVSS is a multiplatform product that offers the same functionalities on Linux and Windows and allows managers running on the two operating system to be interconnected transparently. Although the choice of platform is not completely free due to the usage in the communication layer of some Microsoft-specific communication protocols (e.g., OPC), the compatibility with Linux is very valuable in the scientific environment. PVSS under Linux facilitates the communication with the DAQ, entirely based on the Scientific Linux platform, and allows the shifter to use the same Linux workstation for running both the DAQ and the DCS.

3.3.7 Market and Commercial Aspects

ETM has proven to be a responsive company, accepting to make many changes, some of them major, as a result of CERN's requests. The reliability of the company is of fundamental importance because the product has to be supported for the entire lifetime of the LHC experiments.

3.4 The JCOP Framework

3.4.1 Architecture

The experience gained in the control of the LEP experiments showed that the main problem of the old approach was the lack of standardization. In fact, the usage of heterogeneous programming languages, custom hardware and proprietary protocols led to difficulties in maintaining the software and in integrating the systems under a coherent supervisor. For these reasons, it was decided to rely as much as possible on commercial components (PLC, SCADA Products) and standard protocols, e.g., OPC, DIM. However, the complexity of the task does not allow for the adoption of any off-the-shelf solution. A custom platform is therefore needed to adapt the commercial components to the needs of HEP experiments.

The Joint COntrols Project (JCOP) [22] was started at the end of 1997 with the objective of providing a set of components (the JCOP Framework) to ease the development of control systems for the LHC experiments. The main aim of the Framework Project is to reduce development and maintenance effort and avoid duplication by re-using common components. The framework is also intended to hide the complexity of the underlying PVSS layer and to define some guidelines that ensure the creation of a homogeneous control system both from the point of view of the architecture and of the user interface.

Development tasks are distributed among three roles. The *component developer* extends the framework in order to support new hardware or new functionalities. The *system developer* develops a control application for a particular sub-detector using the tools provided by the framework. System developers are distributed among many autonomous teams (not only CERN-based but also spread all over the globe). The *person responsible for integration* combines the various sub-systems into a coherent control system for the entire experiment.

The layered structure of the JCOP framework (see Fig. 3.3) provides a higher level of abstraction over PVSS. The framework gives the system developers the possibility of using the framework layer but also to access the original PVSS toolkits, if they need to develop specific features not implemented in the framework. To accommodate the different requirements of common facilities, the framework is organized in a series of components, allowing the system developer to choose which packages are required for a particular application.

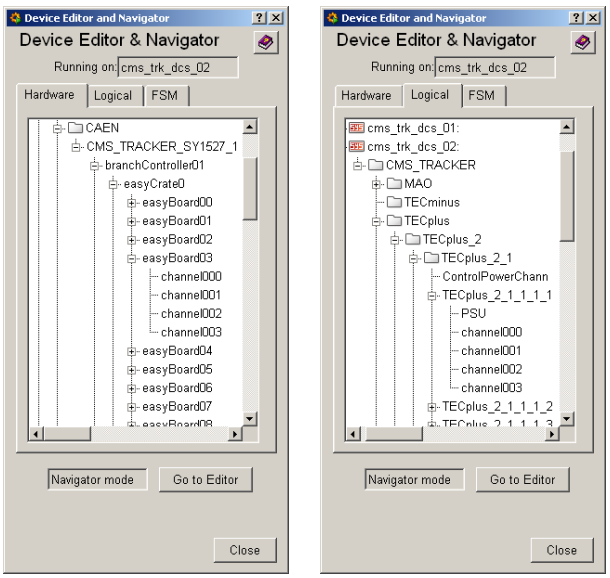

Figure 3.4: Example of hardware and logical view for the power supply system of the CMS Tracker displayed in the Device Editor and Navigator (a tool provided by the framework to the system developer). On the left the hardware view reflecting the structure of the power supply control chain, on the right the logical view that maps each power supply to a part of the detector.

3.4.2 Hardware and Logical View

The framework provides the tools for organizing the controlled devices in a hardware view and a logical view (see Fig. 3.4). The hardware view is a hierarchy that reflects the structure of the auxiliary hardware (power supplies, PLC, etc.) and is not dependent on the correspondence between the auxiliary hardware and the detector hardware. It is coded into the data point name and is usually logically related to the hardware address set in the data point elements. The framework centrally provides the implementation of common device types, as well as a naming convention reflecting the hierarchical organization, e.g., crates that contain boards that contain channels. This prevents different developers from designing different structures for the same hardware, resulting in incompatibilities which would give rise to problems at the time of integration.

The hardware view is mapped to a logical view using aliases for the data point names. The logical view reflects the structure of the detector that is served by the auxiliary hardware. The correspondence between the hardware view and the logical view reflects the physical cabling of the detector.

The logical view typically contains the information of interest to the user. In fact, in order to operate the experiment one is typically interested

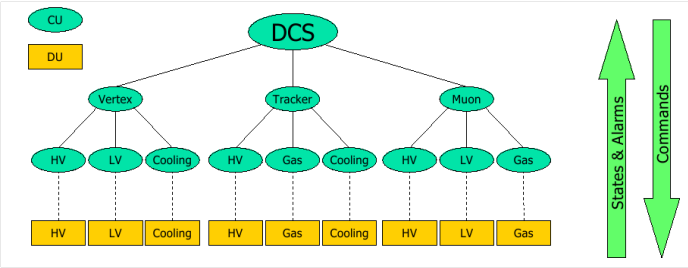

Figure 3.5: Example of a detector tree for a generic HEP experiment. The commands are propagated from the root down to the devices, while the states are reported bottom up. The internal nodes provide an increasing level of abstraction in a status that summarizes the conditions of the hardware controlled within a subtree.

in knowing the part of the detector that is related to a certain piece of information (power supply channel, temperature probe, etc.). Hence, all the user interfaces and geometrical views are usually based on the logical view. On the other hand, the front end hardware experts are interested in the hardware view in order to identify the location of a specific device.

3.4.3 The JCOP Finite State Machine

The JCOP framework provides a Finite State Machine tool, which is the main component for the implementation and the integration of large-scale control systems. This tool combines the functionalities of the SCADA system with the State Management Interface (SMI++) tool [23].

The Finite State Machine concept is used in many different fields with slightly different definitions. In the context of control systems, a Finite State Machine is a model of the system behavior using a finite number of states, transitions between these states, actions and events.

SMI++ supports the hierarchical organization of the Finite State Machine objects, in order to control the experiment at different levels of detail. Typically the parent-child relation reflects a physical container-contained relation but it could also correspond to other kinds of logical relations. Logically related objects are grouped into SMI++ domains. Each domain, which is handled in a separate process, manages the finite state machine logic of its objects and communicates with the SCADA system by using the PVSS API interface. Many SMI++ domains can be connected to cooperate and finally the entire experiment can be controlled by giving commands to a single supervisor node, e.g., the DCS node in figure 3.5.

The leaves of the hierarchical tree correspond to the elementary devices, while the internal nodes provide an increasing abstraction level, up

to the root that represents the entire control system. States and alerts are propagated bottom up whereas commands move in the opposite direction. Control systems are by nature asynchronous. The execution of a command on a device does not change its state directly, but rather triggers an action on the hardware that can asynchronously lead to a state change.

The JCOP FSM [24] provides the developer with three types of software objects:

- *Control Units* (CU) are abstract objects corresponding to internal nodes in the hierarchical tree. CUs run in a separate SMI domain (process)
- *Logical Units* (LU) also represent abstract objects but are running within the process of the CU they are included in
- *Device Units* represent a device and are connected to a data point in PVSS.

An FSM type defines the possible state of the objects and the available actions (commands) in each state.

Device Units represent the bridge between SMI++ and PVSS. Two custom PVSS procedures are associated with every DU type. One PVSS function computes the device state depending on the values of the data point elements that characterize the device. Another custom function is used to translate commands into a sequence of operations on the related data point.

The state of an internal node (CU or LU) is defined on the basis of rules programmed in the type. The rules are of the form

```
when (any child | all children [of type T]
      in_state | not_in_state S)
move_to NewState
```

or

```
when (any child | all children [of type T]
      in_state | not_in_state S)
do Action
```

When a CU/LU object is initialized, it is set to the initial state. The list of the rules in each state is checked in order. The first rule whose condition is verified is executed, leading to a state transition or to the triggering of an automatic action. The actions are implemented as a

sequence of commands of the form `do Action on all children [of type T]` or `move_to NewState`. When an action is propagated to many CUs, they can execute the actions in parallel and report their state back to the parent.

LUs are used at the lower levels, in order to reduce the memory overhead associated with each SMI process. The number of LUs and DUs to be handled in a single CU also impacts the memory usage of the SMI process. For this reason it can be convenient to group strictly related devices in an abstract device handled in a PVSS script (see Sec. 4.3 for details).

The structure of a complex experiment, composed of different specialized sub-detectors and sub-partitions, calls for a flexible partitioning mechanism that allows the users to operate, debug and commission different parts of the experiment in parallel. It is also important to exclude devices known to have a temporary or permanent malfunction from the overall control chain.

Partitioning is related to the concept of *ownership*. An operator can reserve the whole tree or a certain subtree in which case he/she becomes the "owner". Each component has one and only one owner at a time. The subtree can be taken in "exclusive mode" (only the owner can send commands) or in "shared mode" (any other operator with the correct permissions can also send commands). Only the owner can change the exclusivity mode.

Nodes can be included or excluded from the hierarchy. Excluding a child from the hierarchy implies that its state is not taken into account anymore by the parent in its deciding process, the parent does not send commands to it and the owner operator releases ownership so that another operator can work with it. Only the owner can exclude a component from the hierarchy.

Different users can take control of different control units in parallel. In a typical scenario, the shifter excludes a faulty node from the hierarchy and gives the control to an expert, who debugs it and, when the problem is solved, asks the shifter to include the node back in the system.

The FSM hierarchy can be structured to have multiple references to the same node from different parents. When this feature is used, the FSM hierarchy is not a tree but a graph. Multiple references to the same object are useful in many cases to be able to control the same equipment from different views. However multiple references also involve the loss of some important tree properties.

3.4.4 The Conditions Database for Historical Archiving

The Detector Control System is in charge of archiving the changes over time in the system parameters. The archived data is used for analyzing the conditions of the hardware at the time of a problem, to crosscheck the state of the system during physics offline data analysis, or for the control system debugging.

The archiving policy for the CMS control system is to store the data in an Oracle database on a "change basis". This means that the values are not archived by sampling them at constant time intervals, but only when they change by more than a defined deadband. This smoothing strategy is analogous to the driver-level smoothing, but a larger deadband is applied to the archiving in order to limit the number of archived events. The rationale of this methodology is that with a proper choice of the deadband, it is possible to reduce the amount of data to be written to the database. The assumptions for the conditions database is to have an order of 150 PVSS systems and one million parameters per experiment.

Although PVSS comes with an implementation of a database archiver, that is appropriate for standard industrial use, its performance does not satisfy the requirements of the LHC experiments. A collaboration was setup between CERN and ETM (the company that develops PVSS) resulting in major changes in the architecture and several optimizations in order to achieve the needed performance. The improvements in the performance were obtained by optimizing both client side (the PVSS manager communicating to the Oracle database) and server side (the Oracle Database Server). For details on this optimization process, see [25].

3.4.5 The Configuration Database for Changing Running Conditions

The operation of a complex control system implies proper setting of a large number of parameters. Typically the hardware can be configured in various "modes" corresponding to different running conditions (e.g., the temperature limits in the PLC and the alert thresholds must be changed when the set temperature of the cooling system changes). Settings are neither uniform nor constant but can be specific to the individual device (e.g., the current trip limit for a high voltage channel) and may be adjusted by the experts to adapt to changing conditions or to hardware peculiarities.

Moreover, DCS PCs are subject to failures, so that a complete recov-

ery of the functionalities of a PVSS project has to be possible within a reasonable time.

The framework solution to these issues is a tool known as "Configuration DB". This tool handles two kinds of mass configuration data:

Device Static Configuration Data is the configuration data directly related to the configuration of the data points to handle the devices, e.g., the data point names, the address configurations, the data point aliases, etc.

Device Dynamic Configuration Data is the configuration data which may change (relatively) frequently, such as hardware settings (e.g., in the power supplies) or alert limits.

The static configuration is needed when a system has to be completely restored and enables the recreation of the project "as it was before the incident". The runtime database is restored by creating and configuring all the data points according to the static configuration data saved in the configuration database.

The dynamic configuration maintains the parameters to be set in the devices in so-called "recipes". A recipe is described by a set of data point elements and corresponding values and alert configurations. The recipes provide two-dimensional versioning: each recipe is identified by the operational mode and the version number. The operational mode corresponds to a massive change to be applied when basic running conditions change. The version number keeps track of the changes applied over time to the original recipe, to adapt to the evolving conditions of the detector.

The Configuration DB tool uses an Oracle DB as long-term storage system, but also provides a cache mechanism to optimize the performance by reading the recipe from dedicated data points.

Once the parameters are sent to the device by writing to the data points, it is important to check that the changes are effectively applied to the hardware before declaring that the recipe is successfully loaded. In fact, in case of massive configuration of a device, some of the commands may be lost by the driver or in the internal handling of the device. In many cases, it is critical to check that all the settings are correctly applied before switching on the detector, because wrong settings may affect the safe operation of the detector. This is achieved by defining a mapping between the output and the read back element names.

3.4.6 Security Policy for LHC Control Systems

The security policy adopted at CERN to mitigate the risks described in 1.7 is based on a "defense-in-depth" approach where pro-active security

measures are applied to every level (the device itself, the firmware and operating systems, the network connections and protocols, etc.) [26]. To protect the controls network from external threats, the CERN network is separated into various "network domains". The LHC experiments and the accelerator use different dedicated experiment networks.

The interconnection of the experiment network to CERN's General Purpose Network (GPN) cannot be avoided, but traffic crossing the two domains is restricted to minimum. The connection to the GPN is needed for allowing experts to debug the system from remote. Anyway, CERN policy prohibits operating the detector from outside the control room and allows only experts to give commands to the system from remote.

The only means for remote user access are Windows Terminal Servers and Linux-based gateways. For accessing the experiment network from outside CERN, a double login is needed (first into CERN's GPN and then into the experiment network).

Central installation schemes are adopted for control PCs, running Linux or Windows [27]. The central installation ensures the coherent synchronization and monitoring of software versions and system patches on all controlled PCs. However, upgrades and patches are never scheduled automatically. It is up to the system administrator to apply patches and upgrades in a timely manner.

3.4.7 Access Control in the JCOP Framework

The framework implements a user access control mechanism, based on the general CERN user accounts, that works at User Interface level. The framework access control is not meant to protect the control software against malicious attacks, but rather to protect the system from unauthorized non-malicious use (for example, a mistake), to separate the responsibilities among various user roles and to provide traceability of user actions.

The framework access control makes use of the concept of *domain* representing a set of resources and of *privilege levels* that define the level of access that the user can have in that domain.

To ease the definition of the access rights, the group of permissions needed to perform a task is associated to a *role* (e.g., Tracker Operator, Tracker Expert). Users are assigned to one or more roles and can take one of the roles at a time.

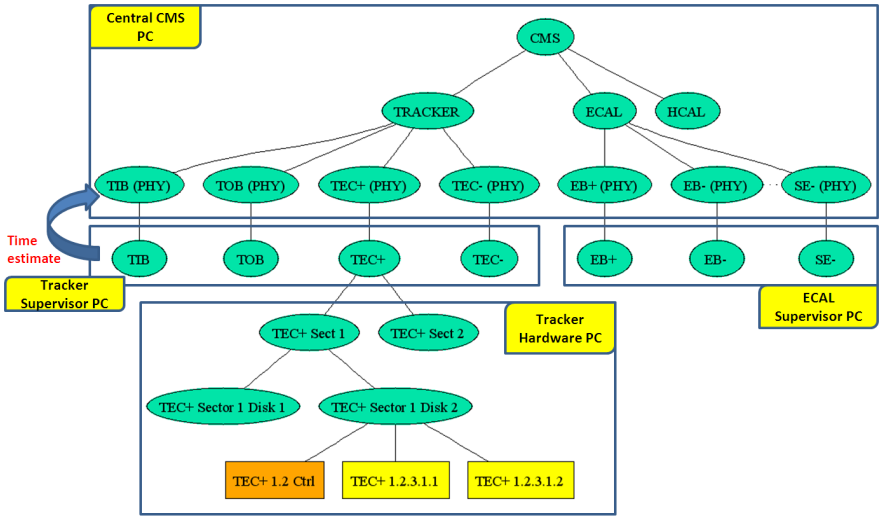

Figure 3.6: The central CMS control tree: the TTC nodes are replicated in the CMS central supervisor in a node with physics-related states. The sub-detector TTC nodes should provide a time estimate for the execution of the command to the central CMS DCS.

3.5 Integration Policies for CMS DCS

In order to ease the integration of the control systems for the various sub-detectors of the CMS experiment, some conventions were provided in the format of an Integration Guidelines document [28].

To integrate the different FSM hierarchies, a central CMS node directly connected to the top nodes of the CMS sub-detectors (TRACKER, ECAL, HCAL, MUON, etc.) was initially proposed. However, the smallest functional unit that can be independently controlled in DAQ is the Trigger Timing and Control (TTC) partition. Each sub-detector is composed of several TTC partitions (for the SST, these are TIB, TOB, TEC+ and TEC-). Hence, it was decided to use the same granularity in the central DCS.

The TTC partitions level is replicated in the CMS supervisor layer (see Fig. 3.6). The objects provided by the sub-detectors must have a common state/command interface, with four predefined states:

ON the TTC partition is ready for data-taking.

STANDBY Ready for beam injection (usually in this state high voltages are off, or set to a low value, and low voltages are on).

OFF All low and high voltages are switched off (used for long shut-down periods).

ERROR A manual intervention is required and no data taking is possible.

Other states can be defined, but are not recognized in the nodes of the CMS supervisor.

When the command to reach one state is given, the TTC node provided by the sub-detector sets a parameter containing a worst-case estimate of the time needed to reach the state. This time depends on the amount of hardware that needs to change state, on the ramping speeds, etc. The parent node starts a countdown and, if the desired state is not reached after the timeout, it moves to an ERROR state.

The RCMS connects to the states of the TTC partition and can pause the data taking, or flag some data as bad, depending on the status reported in DCS. This implies that the partition should change its state only if the sub-detector is no longer ready for physics. Some tolerance may be foreseen: if some small percentage of the detector is not in the proper state, the data taking can still continue and the TTC partition should not change state.

Chapter 4

Strategies for the Implementation of the CMS Tracker Control System

The design and implementation of the DCS for the CMS Tracker Control System is the main subject of this thesis. In this chapter the design philosophy is illustrated and some implementation details are explained. Although the Tracker Control System is completely integrated in the CMS Control System, its complexity and size called for innovative and very high-performing original solutions. Some common strategies and general libraries, originally implemented in the context of the CMS Tracker DCS, were adopted for the DCS of other CMS sub-detectors.

4.1 Principles

The control system for the tracker is designed and implemented following some basic principles derived from experience gained during the design and implementation phases. An advanced prototype of the software was tested in a commissioning run with 25% of the tracker hardware, in 2007. A version providing almost all the final functionalities has been in use since early 2008. These principles proved to be robust enough to cope with the new requirements and with the scaling of the system.

The architecture of the control system is designed to meet three major criteria:

Safety Maximum priority is given to the checking procedures that ensure the correctness of the tracker safety system. Stand-alone scripts can issue protection actions independently from the standard command chain.

Performance Given the complexity in terms of number of devices of the tracker, a special effort is made to optimize the performance for the retrieval of the status of a large number of objects.

Effective user interaction The shifter is provided with the means for fast problem finding; the recovery procedure from error conditions is automated.

The Tracker Control System extensively uses the JCOP FSM tool (see Sec. 3.4.3). The FSM is the main interface used by the shifters for controlling the tracker and provides all the functionalities needed by the non-expert user during normal operation.

The problem of effectively representing the status of such a large system is a major challenge. The power supply status is the main parameter the user is interested in, in order to know if the apparatus is ready for data taking. On the other hand, the status of other sub-systems (PLC, DCU, Cooling System, etc) should be available in the user interface. Feedback is especially needed when the status of a sub-system (e.g., temperatures read by the PLC) prevents a part of the detector from being switched on. The experience with FSM interaction stressed the need to integrate the FSM state with quantitative information and to provide an efficient search method for finding the devices in a certain state. Given the dimensions of the tracker power supply system, a certain percentage of errors in the channels needs to be tolerated in the definition of the FSM states.

Protection actions need to be implemented in DCS. The safety procedures consist of a partial switch-off command given to the power supply system depending on the information coming from various sources. The implementation of protection actions in DCS cannot directly use the FSM to send commands because the partitioning feature has to be circumvented. In fact, a safety action must be executed regardless of the included/excluded state of the devices.

The basic FSM functionalities must be integrated with additional panels and views that reflect other aspects of the system. For example, the expert needs an additional hardware view, in order to commission and debug the power supplies and the PLC system.

A strategy has to be defined to avoid connecting to a large number of parameters from the user panels. For example, it is impossible for the shifter to monitor the trend of all the temperature sensors in the tracker but still it is important to have an effective interface showing the temperature distribution.

On account of these problems, the implementation of the control system follows a common strategy that can be summarized in the following

points. These are analyzed in detail in this chapter. All these issues and the evolution of the relative solutions have been presented in several articles at various international conferences [29, 30, 31, 32, 33, 34].

- The FSM hierarchy is kept simple and mainly reflects the power status of the individual detector elements, the most important information to reach the data taking stage.
- The monitoring of the conditions of the other systems (PLC, DCU, cooling etc) is handled by separate scripts which can react to critical conditions by issuing a (low-level) switch-off command to the power supply system.
- To provide a fast and reliable way to browse the hierarchy, all the static information is cached in a single data point and loaded in memory.
- A custom-developed general algorithm for the propagation of state changes integrates the FSM view with quantitative information. This methodology is completely general and was later adopted by other sub-detectors.
- A custom database was developed in order to represent all the hardware elements, their relations and their internal addresses. This database is used to create the data points and FSM objects. All the information contained in the database is dumped to PVSS where it can be used to query the cabling of the detector.
- The control tasks are distributed as much as possible in different PCs, each one with a specific task and able to handle most of the information locally.
- For some critical parameters (temperatures read from the safety system) the instantaneous mean, maximum and minimum values are propagated in the FSM hierarchy, in order to effectively identify the points with possible problems.
- In case of problems, the user-interaction is minimized by providing a wizard-like procedure for recovery.
- The expert is provided with specific user-interfaces that reflect the structure of the auxiliary hardware (power supplies, PLCs). The usage of a parallel FSM hierarchy was avoided because the hardware view is not needed during the normal operation but only in case of expert operations. Moreover, the commanding feature of the FSM

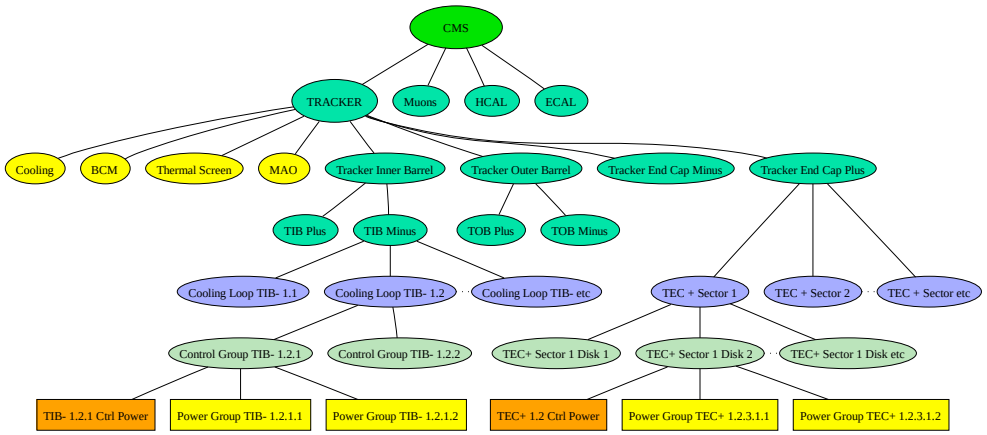

Figure 4.1: Simplified view of the FSM for the control of the Silicon Strip Tracker. The figure shows a partial hierarchy down to some power groups in the Tracker Inner Barrel and in the Tracker End Cap. The main node is divided into four partitions. The software safeties act on the cooling loops (for TIB and TOB) or on the sectors (for the TEC). Cooling loops and sectors are composed of control groups. Each control group has one control channel and several power groups.

is not needed on the hardware view because it is not safe to execute actions based on the hardware structure.

- The user is provided with some tools to plot the distribution of the values of different parameters upon request. A single request avoids the connection to a large number of parameters.

4.2 Finite State Machine Hierarchy

The root of the FSM hierarchy for the control of the tracker is split into four main nodes that correspond to the Trigger Timing and Control (TTC) partitions in the DAQ system: Tracker Inner Barrel (TIB), Tracker Outer Barrel (TOB), TEC+ (Tracker End Cap on $z+$ side) and TEC- (Tracker End Cap on $z-$ side) (see Fig. 4.1). Since these partitions are handled separately in the DAQ, it is important to provide their state at the top level in order to notify the DAQ system when the tracker is ready for data taking. These nodes are directly interfaced to the broader CMS control system, which duplicates them in a higher level "physics-oriented" state (see Sec. 3.5). All the auxiliary hardware, such as the 48 Power Converters (called MAO) and the cooling plant control, is also handled in direct children of the top node.

The next level reflects, for TIB and TOB, the division between $z+$ and $z-$ side followed by a layer of cooling loops, while TEC+ and TEC-

are partitioned into sectors. Below cooling loops and sectors, the control groups represent the state of a control ring, built up from the basic blocks of the power supply system: the control power channel and the power groups (Fig. 4.1). The FSM hierarchy includes 4 TTC partitions, 132 cooling loops or sectors, 356 control groups and 1944 power groups.

The modules of a cooling loop share the cooling lines, so that in case of problems in the cooling system the temperature increases in the entire loop. The software safety switches off a single cooling loop in response to an anomalous value in the environmental probes related to it. In the case of the TEC, control groups belong to more than one cooling loop, so the hierarchy must necessarily go directly from sectors to control groups. For the TEC, the software safety acts at the sector level, just as in the case of the hardware interlock.

The FSM implements the safe sequence for switching on and off the tracker. During the switch-on phase, the control power supply must be on before being able to switch on the power groups belonging to the same control group. Low voltages must also be switched on before high voltages. The switch-off sequence is reversed. HV are switched off first, then the LV and finally the control channels. The FSM does not accept commands that would bring the detector to a state that does not respect the switching procedure. If a forbidden condition is reached for some other reason (e.g., because of a trip that switches off a control channel), then an automatic action switches off the related power supplies (e.g., the power groups served by a tripped control channel).

State	Meaning
OFF	Everything off
ON_LV	LV ON, HV OFF
HVMIXED	LV ON, only one HV ON
HVRAMPING	Some HV ramping up or down
ON	LV and HV ON
HVON_LVOFF	HV ON and LV OFF
ERROR	CAEN error (e.g., trip, over current)
UNPLUGGED	The PSM is unplugged
INTERLOCKED	The PSM is interlocked

Table 4.1: Possible states of a power group

The FSM logic is implemented in several object types. The basic DUs are the power group and the control channel. The possible states for a power group are listed in table 4.1. The states of the upper nodes

summarize the state of the control channels and power groups in the subtree below. These states depend on the percentage of ON channels and are discussed in detail in Section 4.13.5.

4.3 Handling of the Power Groups

Although the power group is a complex object, composed of two LV and two HV channels plus the status information of the A4601H PSU, it is handled as a device unit in the FSM. For this purpose, a custom script, running independently from the FSM, computes the states and handles the execution of the commands for a power group.

The script managing the power groups communicates with the FSM via a custom data point type that provides a simple client-server interface. This data point type includes a `getState` element and a `sendCommand` element. For each power group, a data point of power group type is created and used as an interface between the FSM and the standard data points provided by the framework for the control of the CAEN power supplies. The script to manage the power groups is connected to the elementary devices of each power group and updates the `getState` element accordingly for each power group data point. The state of each power group in the FSM is connected to `getState`. The FSM propagates the commands by writing to `sendCommand`. The script then connects to `sendCommand` and translates the commands into the proper sequence of actions on the power channels.

The solution of grouping the low-level devices into a single DU significantly reduces the memory consumption associated with the handling of the logic in the FSM.

4.4 Task Distribution

The distributed structure of PVSS allows autonomous projects to be interconnected. Thanks to this feature, the Tracker Control System tries to maximize the role distribution among the various PCs. This approach tends to minimize the communication over the network and favors an architecture based on parallel independent processes that handle the local resources of each PC (see Fig. 4.2). By distributing the data among different PCs, exchanging most of the time high level information only, a large amount of data can be handled effectively.

Four PCs are devoted to power supply control, each one communicating with one of the mainframes via a CAEN OPC server that provides an abstraction layer between the mainframe and the PVSS OPC client.

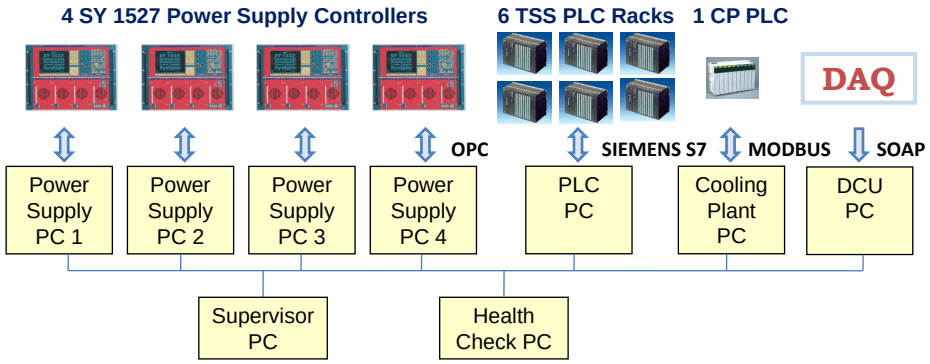

Figure 4.2: The organization of the PCs for the Tracker Control System in the network and the protocols used to communicate with the auxiliary hardware or with the external systems. The power supply mainframes are one-to-one connected to a specific PC. DSS PLCs and the cooling plant PLC are handled in dedicated PCs. The DCUs are transmitted by DAQ to a dedicated PVSS system. Finally a supervisor provides the integration of the different components and specific checking procedures are carried out in yet another PC.

These machines also manage the bottom layers of the FSM hierarchy (from the cooling loop/sector level down to the power group). To improve the performance of the readout, the number of mainframes has later been doubled, connecting each PC to two mainframes (see Sec. 4.9.1).

The cabling scheme of the tracker was adapted to ensure the possibility of connecting the four PCs and the four mainframes one-to-one. In fact, if more than one OPC server connects to the same mainframe (even if subscribing to different items) the performance drops significantly. Moreover, in order to minimize the memory overhead associated with every Control Unit, the power supply channels corresponding to one cooling loop or sector have to be handled in a single Control Unit (corresponding to a separate process running on one machine). To fulfill both requirements, all the power supplies of one cooling loop or sector must be placed in racks controlled by the same mainframe. Moreover, the power supplies connected to one mainframe are all connected to the same side ($z+$ or $z-$) of the tracker (two mainframes per side).

One PC works as supervisor, managing the upper layers of the FSM hierarchy and the connection to the auxiliary systems, such as the beam condition monitoring. All the UI managers are connected to the tracker supervisor PC and run on remote UI workstations. When CMS is operated centrally, however, the UI managers are connected to the central CMS supervisor PC. The supervisor is connected to all other tracker PCs and therefore any data point on any of the other PCs can be accessed from the supervisor.

Communication with external systems (see Sec. 2.4) is handled using DIP, a custom protocol developed at CERN, which provides an effective way of broadcasting information to several distributed locations, using a client/server paradigm.

One PC is devoted to PLC reading from 7 independent PLC racks. The control system takes care of the conversion from ADC counts into physical units (i.e., degrees Celsius for temperatures and percentages for humidities), using probe-specific fitting constants which are read from the custom tracker configuration database (see Sec. 4.7) in the configuration stage. No conversion into physical units is performed in the PLC, because this kind of CPU-consuming procedures would unnecessarily increase the PLC workload. Instead, also the safety limits are configured in the PLC as ADC counts. Any conversion from and to ADC counts is performed in PVSS when needed. The communication with the PLC uses the Siemens-S7 driver, available as PVSS manager. This driver communicates directly with the PLCs without an additional software layer between PVSS and the front end level. A control script in this PC is responsible for updating the mean, minimum and maximum temperatures and to update the state of the critical conditions (see Sec. 4.5 for details).

The PC for controlling the cooling system uses a dedicated PLC to access the cooling plant information. The communication with the cooling plant PLC is achieved via the Modbus protocol. A script in this PC is responsible for updating the cooling plant status presented in the FSM and for taking automatic actions in response to safety critical situations regarding the cooling plant (for example, when the valves of a cooling loop are closed).

The DCU values are managed in a separate machine. In this case the values are not directly read out by PVSS via a driver, but are received from the tracker DAQ system. A dedicated control script computes the mean and maximum values of the various DCU parameters and can issue a switch-off action to the power supply data points (see Sec. 4.6 for details).

4.5 PLC Probes Handling in DCS Software

Cooling pipe or silicon module temperatures are grouped by cooling loop (in TIB and TOB) or sector (in TEC). This grouping is identical to the interlock groups defined in the safety system (see Sec. 2.3.3).

A dedicated script, running in the PC managing the PLCs, computes the instantaneous mean, maximum and minimum values of the probes belonging to each cooling loop/sector. These values are also propagated

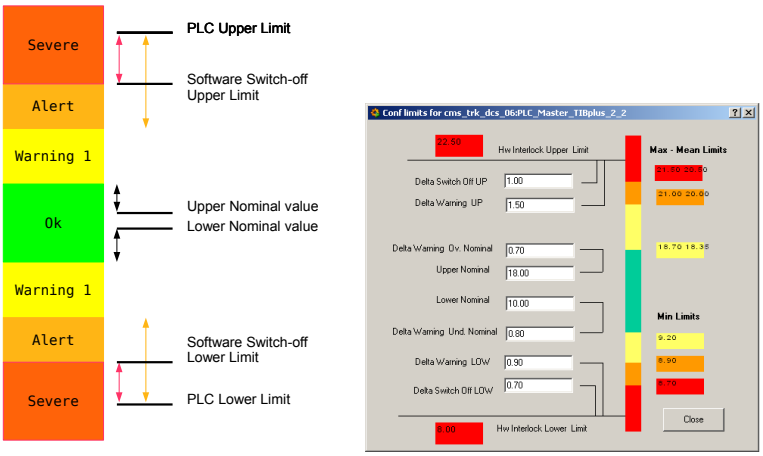

Figure 4.3: Alert ranges for the cooling loop: a set of differences (represented by the arrows) are used to define the alert ranges depending on the nominal temperature and the hardware safety system limits. On the right an example of the configuration for a TIB cooling loop in warm mode.

to the upper levels in the FSM hierarchy using the propagation strategy described in 4.13.

Each PLC probe has two hardware interlock limits, a maximum and a minimum (see Sec. 2.3.3). The software configuration defines software switch-off limits that are narrower than the hardware ones (software maximum below hardware maximum and software minimum above hardware minimum). In addition, the software defines alert ranges of increasing severity to signal an anomalous temperature value. Acceptable temperatures are defined in terms of a nominal range. These ranges are configurable for each cooling loop or sector. It may be noted however that the absolute software limit values are not explicitly configured. Instead, the configuration specifies relative limits in terms of the difference between the hardware limit and the corresponding software limits. In this way the software limits are recomputed automatically each time the hardware configuration is updated. This ensures that the software limits are narrower than the hardware interlock limits. For a schematic representation of the alert ranges defined at the cooling loop level and their relation with the hardware interlock limits, see Figure 4.3.

The temperatures across a cooling loop are fairly uniform, so the probe readings are expected to be within a few degrees of each other. In the case of cooling loops with many probes, the probes give somewhat redundant information. The safety mechanism therefore ignores a single inconsistent reading, a mismeasurement given by a noisy probe. However, if two probes in the same group give a consistent out-of-limit

measurement, the safety mechanism is triggered. Finally, there is an additional software limit for the mean value. When the mean value of the probes in a cooling loop falls outside a specified range, the protection action is triggered. The software protection also prevents switching on a cooling loop until the temperature values are within the normal range (see Sec. 4.12 for details).

The hardware interlock acts with a coarser granularity than the software safety. It interlocks one or more crates instead of several individual power supplies (see Sec. 2.3.3). The software safety is expected to intervene before the hardware interlock limit is reached. When the cooling loop is switched off, the temperature decreases and goes back into the safe range, preventing the interlock from switching off other parts of the detector as a side-effect.

Dew point values are computed from pairs of related air temperature and humidity sensors. The dew points are compared to the coldest point in the corresponding region of the tracker. An alert is raised when a dew point gets too close to the coldest temperature in the region. In case a critical threshold is reached, the corresponding sub-detector is switched off and prevented from being switched on again.

If an interlock is fired by the DSS system despite the implemented software safeties, the DCS must send a double hardware acknowledge before the tracker can be powered again. When the values are back in the proper range, the single probes that went out of limits must be acknowledged in the PLC system to release the interlock. Then the power supply system has to be cleared with a command given at the level of the mainframe and the power can be switched on again. The analysis of which probes must be acknowledged is quite complex and must be somehow automated. This functionality is integrated in a special recovery wizard (see Sec. 4.14).

Communication from the control system to the safety system during normal operation is limited to acknowledge signals. However, two kinds of configuration procedures are foreseen. The first, used in case of change of the running conditions (for example, when the set temperature of the cooling plants is changed), updates the hardware limits for the PLC probes. In this case, limits are downloaded to the PLC (as ADC counts) and the software limits are recomputed according to the software configuration. The second procedure is the configuration of the interlock groups. This is a very safety-critical operation that is performed by the experts only in case the PLC should be reconfigured from scratch.

The initialization of any configuration should be made very secure and restricted to experts only. For this purpose an access control mechanism is in place. Moreover, to enable configuration of PLCs only from reliable

sources, the communication from the control system to the safety system must follow a well-defined hand-shaking protocol.

At configuration time, the control system reads back at each step the last written values to be sure that the writing procedure worked correctly. After receiving any configuration from PVSS, the PLC performs some internal checks. If the configuration is considered to be safe it is accepted, otherwise the PLC keeps the old configuration. The new configuration is not accepted, for example, if the communication protocol is not followed correctly or if the number of participating sensors is too small or if no interlock groups are defined. The settings of each probe (limits, enable flags, etc.) are read back by the DCS from a different memory location than the one that is accessible for writing at configuration time. The PLC copies the configured values to the final location used by the PLC logic only in case of successful configuration.

4.6 Handling of the DCUs and Communication with the DAQ

DCUs (see Sec. 2.3.4) provide environmental information with the granularity of the individual module. The DCUs include several types of environmental sensors measuring temperatures at various points, voltages, currents and humidities. This information should naturally be treated in DCS that is responsible for displaying and archiving the values read from the DCUs and for executing protective actions in response to any critical condition. However, the DCUs are read out by the DAQ system via the data acquisition control chain. Hence, a communication between DAQ and DCS is needed. In order to send data to DCS, DAQ uses PVSS SOAP eXchange (PSX), a server that provides a Simple Object Access Protocol (SOAP) service that allows external applications to communicate with PVSS using this standard protocol.

When the front end electronics on the modules are switched on, a DAQ process reads the DCU values periodically and independently for each TTC Partition. The final user is not interested in the raw data of the DCUs and the DAQ database contains the DCU-specific conversion constants. Hence, the conversion into engineering units is performed in DAQ, and DCS receives the converted values.

After the DAQ has sent all the DCU data, it sends an acknowledge command to the control system by writing to a predefined data point element. The update of this element causes the recomputing of the mean and maximum values of the DCU elements related to each power group or control group. This strategy is needed to avoid unnecessarily recom-

puting these values each time a DCU is updated and to take advantage of the fact that data is sent in bunches. DAQ implements a smoothing procedure, by keeping track of the latest values and sending the new data only in case the difference exceeds a defined threshold.

For each DCU parameter, limits can be defined for both the mean and the maximum value at the level of power group (for front end DCUs) or control group (for DCUs on CCU). When these limits are reached, a safety procedure switches off the related power supplies. Since the information coming from the temperature readings in the DCUs has finer granularity than the information read from the PLC system, the protection action based on DCU values should in principle prevent the intervention of the next software safety level.

The information from DCUs is only available when the modules are powered, so after a safety switch-off it is not possible to get the updated values from the DCUs. For this reason, the DCU values do not inhibit the switching on of the power supplies after a software switch-off.

4.7 The Custom Configuration Database

The creation of the tracker logical and FSM hierarchy requires the mapping from the hardware (e.g., power supply units and environmental probes) to the corresponding set of silicon strip modules (e.g., power group, control group, cooling loop). This information is also needed to configure the interlock logic in the PLC. Moreover, the correctness of the map is crucial for operation and safety and must be carefully checked.

In order to have a coherent data source describing the tracker structure, information coming from various and heterogeneous sources has been assembled into a custom designed Oracle database. This database holds the representation of all the nodes of the FSM hierarchy and the cabling and configuration parameters of the hardware. Because of the size and the extreme complexity of the tracker hardware, it would not have been possible to build the required components without the help of a database. The existing information sources were insufficient, typically stored in inadequate formats (simple spreadsheets) and did not provide the correlation between the different hardware subsystems.

The usage of a relational database has the advantage that data can be split into different tables, using foreign keys to represent the relations between different hardware. The information can then be combined in several views that are used to configure all the PVSS projects. Unique constraints and primary keys provide some first checking procedures on the data. For example, the constraints can automatically check that two

power groups are not connected to the same power supply unit.

Det.	a	b	c	d	e	f
TIB (1)	End	Layer	Cool Loop	Ctrl Ring	String	
TID (2)	End	Cool Loop	Disk	Ctrl Ring	String	
TEC (3)	End	Sector	Disk	Cool Loop	F/B	PG
TOB (4)	End	Layer	Cool Loop	Ctrl Ring	Rod	

Det.	a	b	c	d	e	f	Meaning
0	0	0	0	0	0	0	Root (Tracker)
4	0	0	0	0	0	0	TOB
2	2	0	0	0	0	0	TID minus
3	1	2	1	1	1	1	TEC plus Sector 2 Disk 1 1.1.1

Table 4.2: Meaning of the digits in the database key and examples of encoded names. The convention can be used to define both the internal nodes and the devices.

To ensure a homogeneous naming convention, each detector part has a unique 7 digit identifier. The digits have different meanings depending on the sub-detector, identified by the first digit. By using a 0 in the columns that do not need to be specified, the same convention can be used for both the lowest level devices (power groups) and the internal nodes (Table 4.2). The FSM hierarchical tree is represented in a table with a self-reference, pointing for each node to its parent (see Fig. 4.4). The same representation can be extended to identify the PLC probes by adding two additional digits, since the probes are logically related to a power group or a control group. The two additional digits are needed because in some cases a power group can host more than one probe.

The configuration of the interlock groups is an example where the usage of the database to remove the tight coupling of data is very helpful. The sensor and relay bitmasks for an interlock group (see Sec. 2.3.3) must be specified in terms of the PLC internal addresses of sensors and relays. In the database, it is sufficient to specify the relation between logical names of the probes and the interlock groups (to define which probes belong to the group) and between the relays and the crates. The database automatically computes the bitmasks by finding the internal address of the probes and of the relays connected to the crates that must be interlocked (see Fig. 4.5). In this way, if a mistake is found or a change in the cabling scheme is introduced, it is sufficient to correct the information at a single point in the database, and all the related views are automatically updated to reflect the changes.

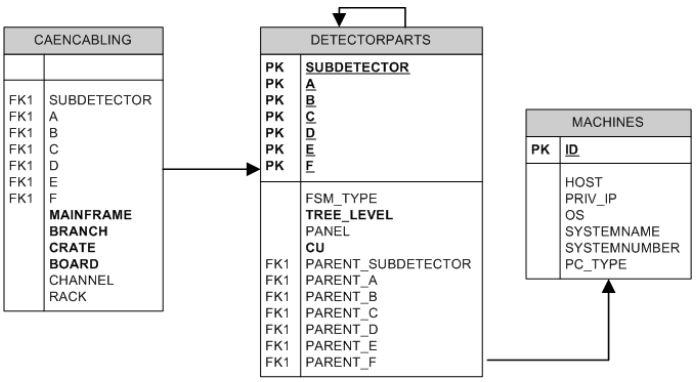

Figure 4.4: UML diagram for the tables of the custom configuration database describing the FSM hierarchy. The main table DETECTORPARTS has a self reference, to model the child-parent relation. The reference to the machine table is used to identify the PC where the FSM node must be created. The table CAENCABLING gives the position of the Power Supply Unit (or channel) for the power groups and the control channels.

The database furthermore facilitates the development of the tools implementing the checking procedures (see Sec. 4.8). During standard operation the connection to the custom configuration database is not needed since all relevant information is available in PVSS (in the FSM hierarchy or explicitly dumped to special configuration data points).

The custom configuration database is used for creating all the PVSS projects used for the Tracker Control System from scratch. After complete configuration of the system, an image of each PVSS project is saved to the general configuration database provided by the JCOP framework (see Sec. 3.4.5).

4.8 Checking Procedures for the Configuration of the Safety System

To ensure the correctness of the cabling, some checking procedures have been performed. These procedures ensure that the real cabling of the detector is consistent with the information contained in the custom configuration database. In fact, the cabling of the CMS tracker involves the connection of 2300 power cables (1944 LIC cables connecting the power groups and 356 PLCC cables connecting the control rings). The cabling of the detector is split into two segments (see Sec. 2.3.1). Hence, there is a total of 4600 power connection points, where potential mistakes could be made.

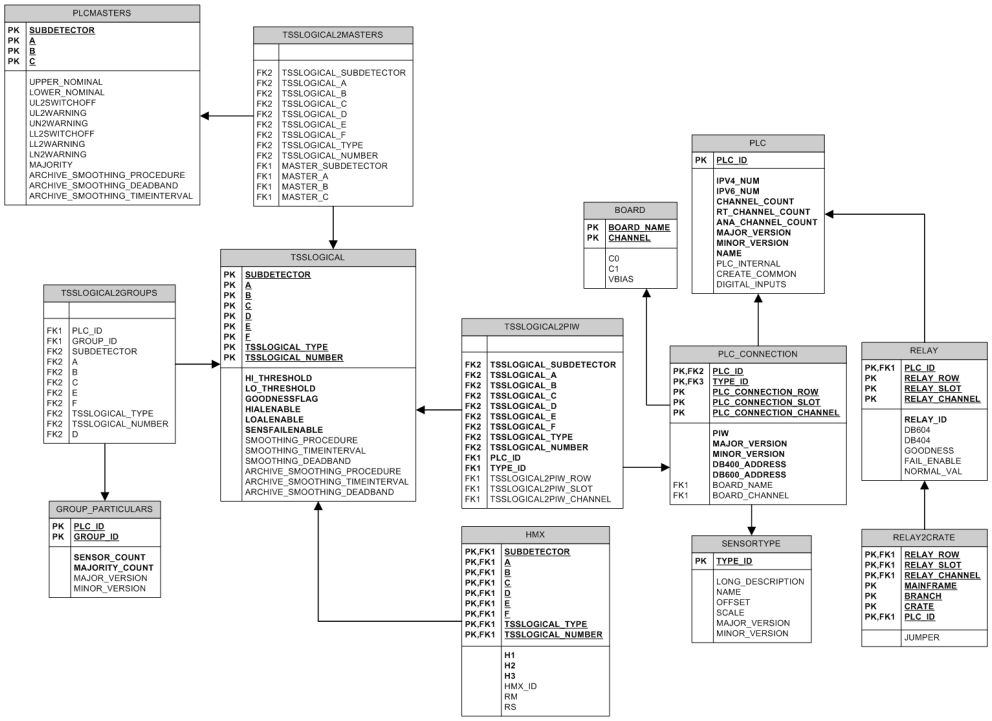

Figure 4.5: UML diagram for the tables of the custom configuration database describing the PLC connection. The main table is TSSLOGICAL, which represents the environmental probes (identified with the same naming convention used for the detector parts). The probes are mapped to a PLC master, representing the cooling loop or sector they belong to (table PLCMASTERS), and to the interlock groups (table GROUP_PARTICULARS). The internal address of the probe is found in two steps: the logical probe is mapped to the row, slot and channel that identifies the connection to the PLC rack, then each position (row, slot and channel) is mapped to the internal PLC address (table PLC_CONNECTION). Some types of sensors require additional conversion constants that can belong to the conditioning board (table BOARD) or, in the case of the HMX sensors, to the probe (table HMX). The table PLC represents the PLC rack (TIBplus, TIBminus, etc). The relays are listed in the RELAY table and their connection to the crates is specified in the RELAY2CRATE table. In order to compute the sensor and relay mask, the information contained in these tables is combined with the cabling of the power supplies (see Fig. 4.4) to identify the relays that must be fired.

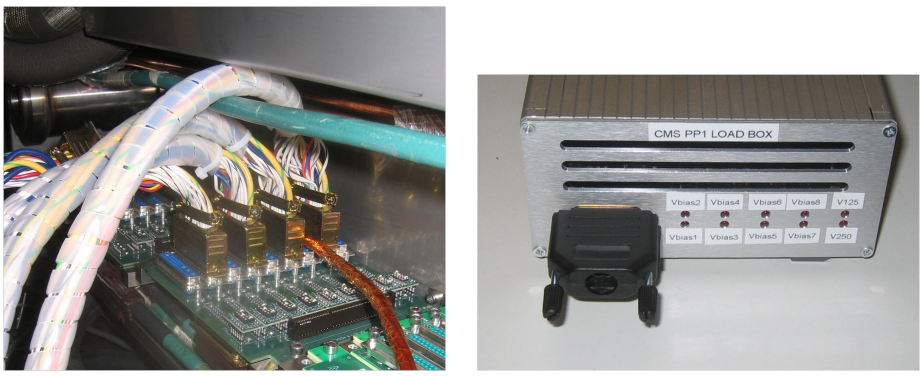

Figure 4.6: The cables for the checkout connected to the PP1 connection board (left) and the load box used for the checkout (right).

When a small section of the tracker is not powered, the related modules are completely blind for track reconstruction. Since the detector hardware is somehow redundant, track reconstruction may still be possible, provided that the correct information about the availability of each part of the detector at a certain time is correctly recorded. The conditions database provides the historical status of the power supply system. The correctness of the mapping is therefore crucial for offline analysis.

Moreover, in the worst case a wrong cabling can compromise the safety, since the hardware interlock would not switch off the power supply connected to the correct detector region in case of a critical condition. Though the Tracker Safety System is completely independent of the control system during normal running, ensuring its correctness is a crucial task which must be performed by the control system during commissioning.

The first checking procedure was the PP1 checkout. This phase was performed before the tracker was inserted into CMS. During the checkout, all the PP1 locations were connected to custom load boxes (see Fig. 4.6). Since the readings of the environmental probes are also transmitted via the power supply cable, the procedure also checked that the signal is reported to the PLC in the expected memory address by using special simulators for the temperature and humidity sensors.

A specific program, directly usable on a laptop by the operators performing the test in the UXC, guided the checking procedure. The usage of a load box with different resistances served to test several PP1 connectors in parallel. The results of the checks were stored in a database in order to keep track of the errors.

After the successful result of the PP1 checkout, the tracker was inserted inside CMS and the last segments of the power supply cables,

linking the PP1 to the detector, were connected.

The correctness of this last connection was checked by powering the power groups one by one and verifying that the unique identifiers of the DCUs read out by the DAQ system matched the ones listed in the Tracker Construction Database. This special procedure was managed by the DAQ system, giving commands to the DCS via the PSX interface.

The last important check performed was the test of the interlock safety. Once the correct connection between the power supplies and the tracker has been checked, it is critical to verify the mapping between the PLC probes and the crates interlocked by the PLC relays as a result of one or more out-of-limit probes. This safety-critical test is done by changing one hardware limit at a time for all the probes that can cause an interlock and then verifying that the proper crates are interlocked. When two probes are required to fire an interlock, the program checks all the pairs of consecutive probes, to be sure that every probe that is supposed to be in the interlock group actually fires the interlock. This procedure must be repeated after each reconfiguration of the interlock groups in the PLC. A fast version of the checking procedure, checking randomly two cooling loops per PLC system and changing the limits for the needed number of random probes, is also in place.

4.9 Performance Analysis of the Communication with the Hardware

4.9.1 Performance of the Communication with the Power Supply System

The communication between DCS and the CAEN power supply system (see Sec. 2.3.1) uses the OPC protocol. PVSS provides a generic OPC client while CAEN provides an OPC server that can be interfaced to the SY1527 mainframe. The client subscribes to some items in the server and groups them in OPC groups that share various parameters, notably the refresh time.

The monitored parameters can be divided into three classes, read out with different polling times. Three types of OPC groups with different timings are defined. Fast groups handle crucial items (channel status, monitored voltages and currents), slow groups are used for read back of user settings and very slow groups are used for items that change very rarely, such as the board serial numbers.

From preliminary tests it became clear that better performance is achieved with sufficiently small OPC groups. When a single OPC group

(or three groups with different polling rates) was used for all the items of the same mainframe, the performance was very bad and in some cases led to the impossibility of reading out some items.

Hence two different grouping strategies were tested: three OPC groups per crate or three OPC groups per rack (branch controller). The tests were performed in an early development stage when neither the mainframe firmware nor the CAEN OPC server had reached a final version. However, there are reasons to believe that the choice of the optimal size is still valid with the new software. The resulting distribution of the response time for the state change of one channel in the two different configurations is shown in Figure 4.7. It is clear that the solution to group the items by crate leads to shorter response times.

Another variable that can be scanned is the polling time of the fast reading group. It was observed that decreasing the refresh time below a certain threshold (5 s) does not yield a faster response.

Response time depends of course also on the number of active OPC groups and increases significantly when more than 20 groups are active (Fig. 4.8).

The tracker power supply setup was a unique opportunity to perform these extensive performance tests. In fact the required hardware was only available at Point 5 and the relevant tests could not be performed by CAEN.

While the grouping strategy (decided on the basis of this early data) improved the communication time, the achieved performance was still not satisfactory. The response time for switching on or off one single channel was still longer than 10 seconds, an unacceptable time for any user to receive feedback for a command. The main problem was that the polling strategy adopted in the communication between the OPC server and the mainframe forced the reading of the complete information at every readout cycle. The unsatisfactory performance of the readout is due to the fact that the mainframe, originally designed to handle up to 16 boards with a maximum of 8 channels each, is used, under the EASY scheme (see Sec. 2.3.1), to control up to 16 branch controllers, with up to 450 channels per branch controller. The results presented in this paragraph clearly show the poor scalability of the polling based communication.

A request for an improved performance was sent to CAEN and led to a new implementation of the communication between the OPC server and the mainframe, based on an event-driven strategy.

In PVSS, communication between the Event Manager and the other types of manager (drivers, control managers, UI managers) is event-driven. Nevertheless, PVSS managers are not the only components used

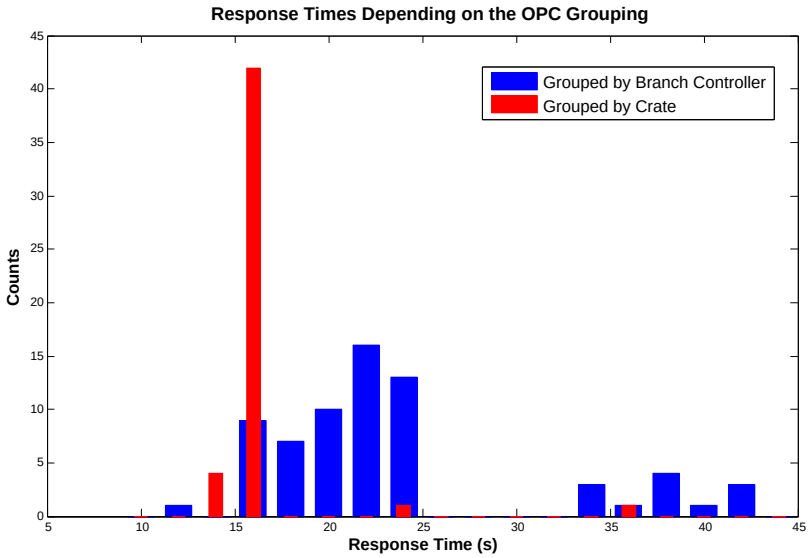

Figure 4.7: Distribution of the response time for switching on a channel with two different configurations of the OPC groups (by branch controller and by crate). By grouping by crate, a significantly lower response time is obtained.

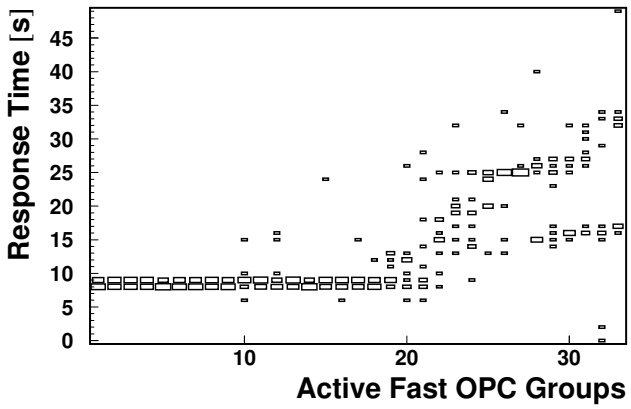

Figure 4.8: Distribution of the response time depending on the number of active OPC groups)

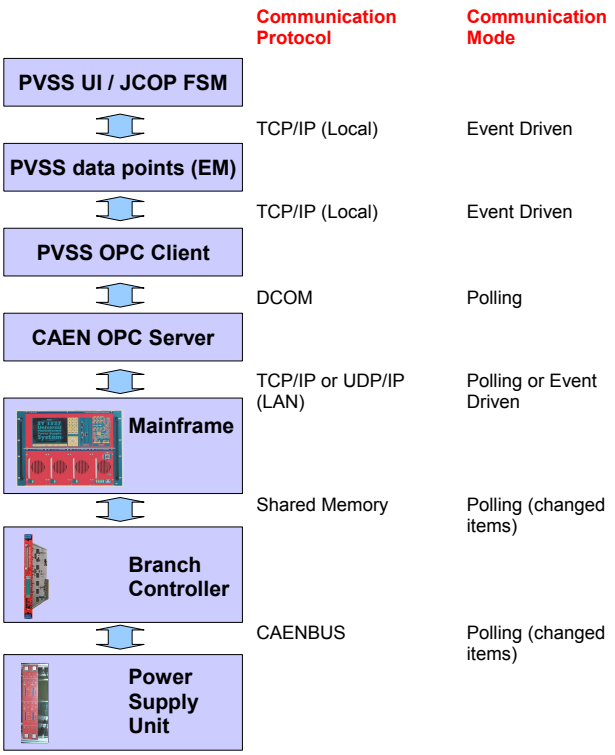

Figure 4.9: Communications stack for the control of the CAEN PSMs. The upper levels are handled in PVSS and follow an event driven protocol. Communication between OPC client and server always uses a polling mechanism. The new version of the OPC server provided by CAEN offers the possibility to communicate with the SY1527 mainframe via an event-driven protocol. Communication between the mainframe and the power supply units is done in two steps and uses a polling mechanism (but only for the items that have changed since the last cycle).

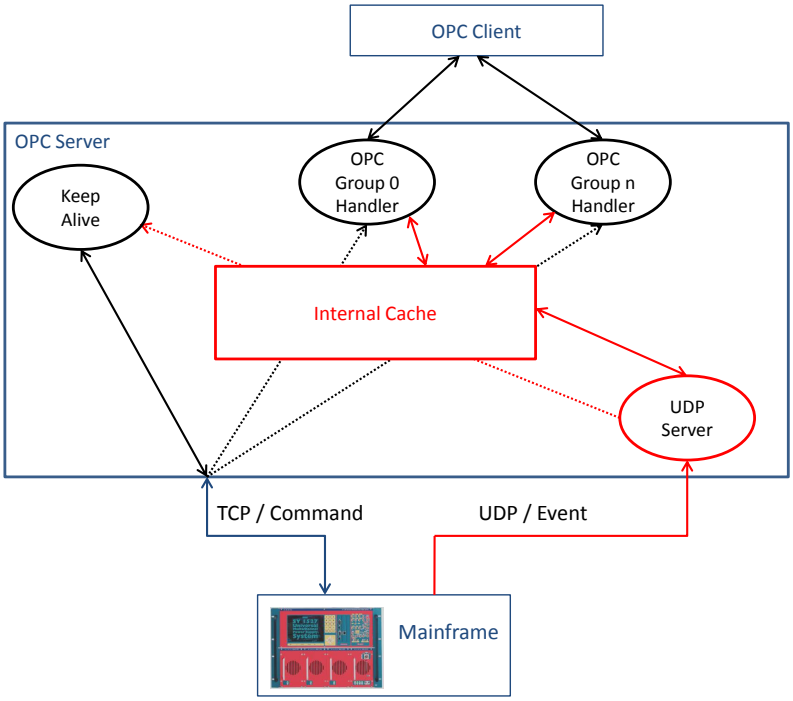

Figure 4.10: Software architecture of the event-driven version of the CAEN OPC server. When an OPC client creates an OPC group, it initiates a thread inside the CAEN OPC server that polls the data from the internal cache. The OPC server subscribes to the relevant items in the mainframe. A separate thread (UDP server) is responsible for updating the internal cache of the server when it is notified of a change by the mainframe. The commands are still sent via TCP. The new features (implemented following the request of the tracker) are represented in red.

in the communication with the devices. The communications stack for the control of the CAEN power supplies is sketched in Figure 4.9 where all the relevant software layers are represented together with the corresponding communication protocol and communication mode.

The upper layers (User Interface, Event Manager and OPC client) are handled by PVSS and use an event-driven protocol. The communication between the PVSS OPC client and the CAEN OPC server uses a polling mechanism, based on the refresh times defined in the OPC groups. The new features introduced by CAEN (following the request from the CMS tracker) changed the communication mode between the CAEN OPC server and the CAEN mainframe. The new implementation required changes both in the OPC server and in the mainframe firmware.

When the event-driven strategy is enabled, the CAEN OPC server keeps the image of the OPC items in memory. This cache is updated by the mainframe on a change basis (see Fig. 4.10). The event-driven mode uses UDP instead of TCP as transport protocol. Hence there is no concept of acknowledgment, retransmission and timeout (some messages may be lost at the network layer), and the order in which the messages arrive cannot be guaranteed. To avoid inconsistencies, if some items are never received within a configurable verify time, these items are read again (using polling) through TCP/IP. UDP was preferred over TCP essentially for performance reasons. However, CERN IT department is currently working with CAEN on the optimization of the event-driven protocol. The optimized version should reduce the traffic by grouping several updates in a single packet. This way it should be possible to implement the event-driven protocol over TCP/IP.

The event mode is transparent for the OPC client, that still queries the server with the specified polling time and receives the data from the server cache. The event-driven strategy can only be used for data read from the mainframe, while a parallel connection using the TCP protocol manages the commands to the mainframe.

Communication from the mainframe to the PSUs is done in two steps. The mainframe reads the state from the branch controller and each branch controller queries independently up to 6 crates.

The mainframe firmware is responsible for providing the interface between the branch controllers and the OPC Server. Hence, the OPC Server can be regarded as a client of the mainframe firmware. The mainframe keeps an internal cache, containing the complete image of all the items exported by the Power Supply Units. This cache is refreshed every 0.5 s by reading the changed items from all the branch controllers. If the communication takes more than 0.5 s, the process misses one or more rounds and restarts to communicate with the branch controllers at the

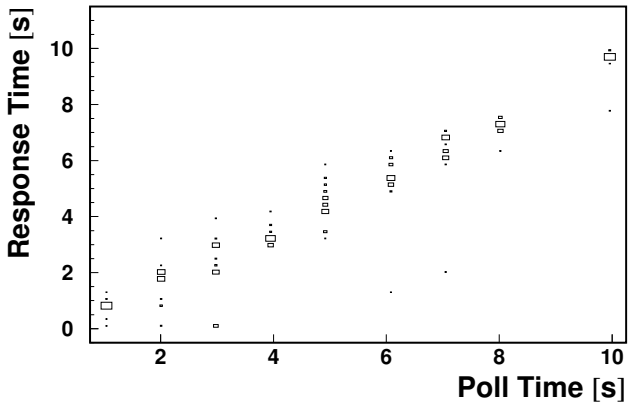

Figure 4.11: Distribution of the response time for setting a parameter depending on the polling time of the group when the event mode communication is enabled in the OPC server. The response time is limited by the polling time.

next time frame. Once the values are updated in the mainframe's cache, a separate process, called Event Dispatcher, is notified. The dispatcher keeps a list of all the OPC Servers that have subscribed to each item and sends the updated values to those that have subscribed to it. In the tracker setup, only one OPC Server is connected to each mainframe, so the event dispatcher handles a single client.

The branch controllers are able to communicate in parallel to up to 6 remote EASY crates via 6 independent communication lines. All the line threads continuously query the remote boards to know which items have been changed. A shared memory is updated with the received data. CAEN claims that the branch controller can refresh about 300 items per crate in 0.5 s. In the case of the tracker, the crates are completely full and about 500 items have to be handled, so that a complete refresh of the status of one branch controller should take around one second in the worst case. However, only the items that have changed are read out, so most of the time the status of a branch controller should be refreshed in 0.5 s .

Unfortunately it was not possible to extensively test the response time for switching on one channel depending on the polling time with the new event mode strategy because of the unavailability of the hardware. However, some tests were made of the response time for changing the value of one parameter depending on the poll time.

Figure 4.11 shows the results. When the event-driven mode is used, the response time increases linearly with the polling time that acts roughly as an upper limit for the response time. The great impact of the event-driven communication is evident in the figure. In this case the updated

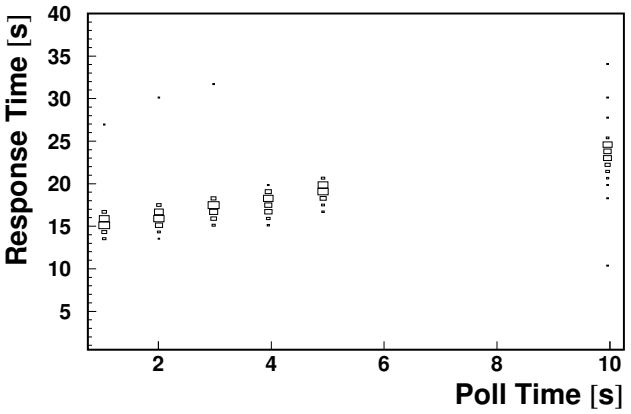

Figure 4.12: Distribution of the response time for setting a parameter depending on the polling time of the group when the communication between the OPC server and the OPC server uses polling. In this case the response time never drops below 15 seconds.

value is retrieved most of the times during the next polling after the setting. For comparison, the same test was performed on a similar setup where the polling mechanism was used (see Fig. 4.12). In this other case the mean response time was above 15 s, regardless of the polling time. However, this very good performance is only observed when the total number of changes is low. During a massive state change (for example, when the entire tracker is interlocked) the performance of the event mode drops drastically. This effect should be mitigated by grouping several events in the same communication packet.

Based on these results, polling times of 2 seconds (for fast groups) and 10 seconds (for slow groups) were used, allowing the OPC client to retrieve the needed information with a good feedback time during stable conditions.

An alternative solution for improving the performance of the communication is to relieve the load of the mainframes by doubling the number of mainframes and halving the number of branch controllers per mainframe. This strategy does not require any recabling of the power supplies and was finally successfully adopted in the tracker setup (that is currently using 8 mainframes). The response times for the new setup are acceptable also when using the polling based communication method.

4.9.2 Performance of the S7 Driver in the Communication with the PLCs

For optimizing the communication with the S7 driver, used to communicate with the PLCs of the Tracker Safety System, the polled items are split into three groups, read out using different polling times. The 1200 sensor readings are assigned to a fast group with a refresh time of 2 seconds. The state of the PLC relays is read out every 5 seconds, giving a total of 544 items. The other parameters are read at a lower rate (about 20 seconds). A smoothing is applied at the driver level to reduce the number of events in the event manager. However, fast values are read every ~ 3 minutes even if they have not changed, using slightly different time thresholds to avoid a synchronous mass request. This solution increases the confidence of the user in the displayed data even in case of stable temperatures.

Increasing the number of polling groups for the S7 driver does not achieve a better performance, so a single poll group is used for each different speed.

4.10 Performance of PVSS dpGet and dpSet

The functions `dpGet` and `dpSet` are the basic commands for the interaction between a user interface or script and the PVSS Event Manager. Both commands can be used to get or set many elements in a single function call, provided that all the data point elements belong to the same PVSS system. This limitation is due to the fact that each `dpGet` or `dpSet` issues a request to a specific event manager.

Commands acting on data points always imply an overhead due to the connection to the central manager. For optimizing performance, a good practice is to avoid the calls to `dpGet` and `dpSet` as much as possible if the operation can be directly implemented in the manager's local memory. It is also useful to group several data point requests in a single call.

As a rough guide, the data not read directly via drivers should be saved in data points only if it needs to be visualized, accessed from various concurrent scripts or archived to the historical database. The intermediate data that needs to be accessed frequently should instead be kept in memory. When the data contained in some data point elements has to be accessed for reading by different concurrent threads in the same control script (that share the same memory), it is recommended to access the data from a mirror in memory. This mirror is kept actual by connecting to all the elements and updating the value in memory. This approach is particularly advantageous when the mirrored elements are

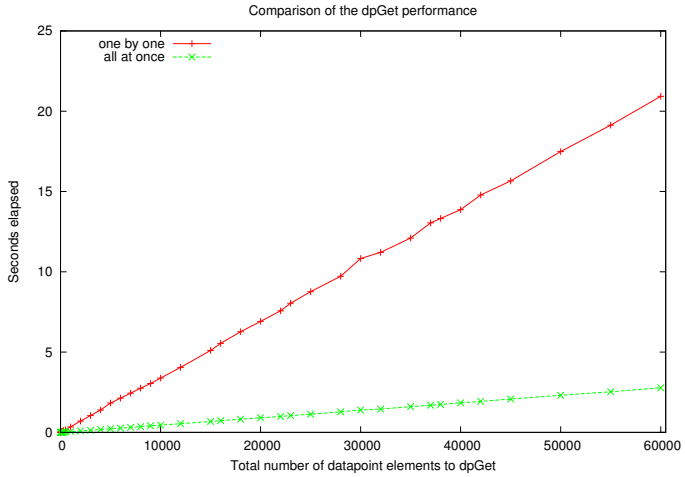

Figure 4.13: Comparison of elapsed time for getting an increasing number of data point elements one by one or with a single request

changing much less frequently than they are accessed.

In order to investigate the performance of these two fundamental functions, an extensive set of tests was performed. All the measurement are referred to a PC equipped with dual core 1.8 GHz Intel processor and 2 GB of RAM.

Plot 4.13 shows that the time needed to perform a `dpGet` command scales linearly with the number of retrieved elements. When one single request is issued for each data point element the overhead time (about 0.3 ms per request) is dominant. By grouping the requests in a single command, the response time is kept within an acceptable range. For example, for 5 000 data point elements (which is a reasonable number in many applications) the grouping strategy improves the elapsed time from 1.8 to 0.2 seconds.

The behavior of `dpSet` is different. The time taken for setting the elements one by one increases linearly with the number of attributes to be set, whereas the time taken to set the elements with a single command appears to increase as $\sim N^2$. The quadratic behavior is shown in figure 4.14, which compares the times needed to set an increasing number of data point elements one by one or with a single call. The point where setting one element at a time becomes faster than setting the elements in a single request is around 8 000 elements.

This quadratic behavior is unexpected and the underlying reasons for this poor performance are not fully understood. The issue was discussed with the company. However, the problem is still not fixed and a workaround is needed at JCOP framework level.

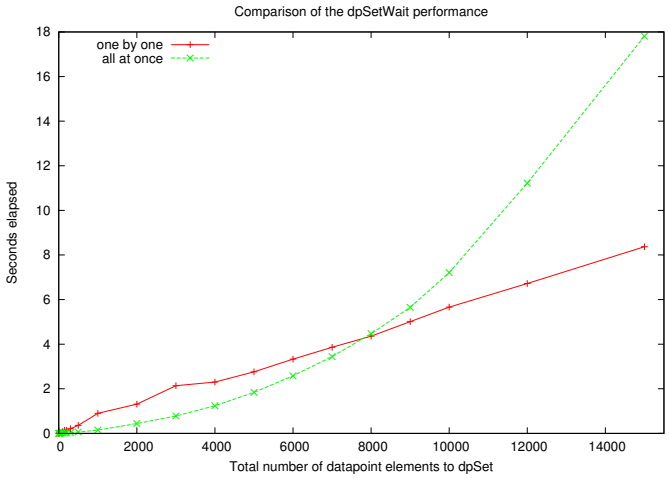

Figure 4.14: Comparison of elapsed time for setting an increasing number of data point elements (up to 15 000) one by one or with a single request

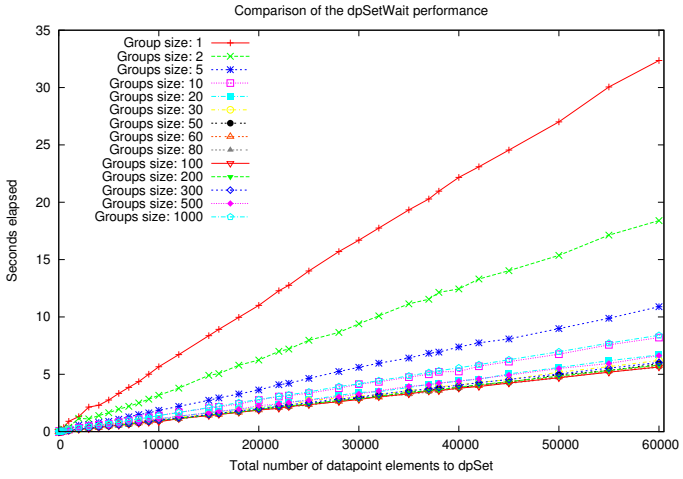

Figure 4.15: Comparison of elapsed time for setting an increasing number of data point elements (up to 60 000) varying the number of requests and the group size (i.e., the number of elements per each request)

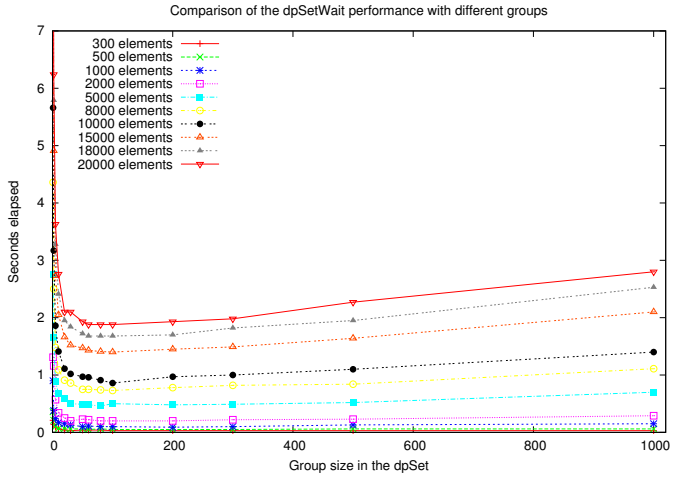

Figure 4.16: Dependence of the elapsed time for setting a constant number of data point elements on the size of the request. The time taken to set the elements decreases when the group size goes from 1 to 100 then slowly increases again.

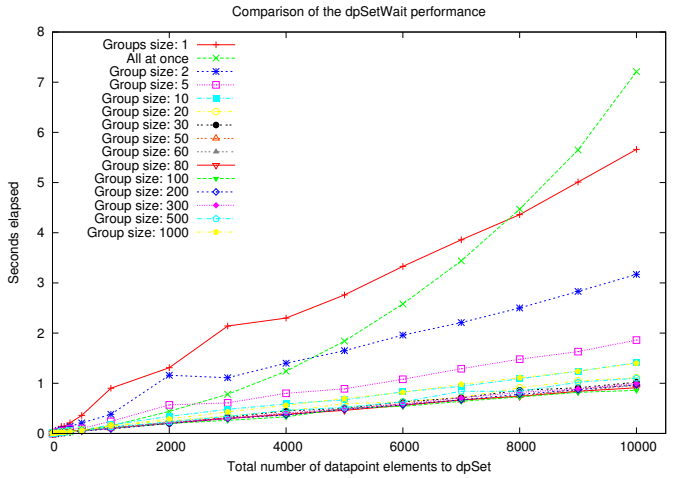

Figure 4.17: Detail of smaller values (up to 10 000) for groups of different sizes

A study was performed to determine the optimal solution for writing a large number of attributes. It is clear that grouping the elements in small groups helps because it is desirable to stay in the region where the quadratic behavior is performing better than the linear one. A set of measurements was taken for a large number of data point elements (up to 60 000) to determine the optimum size of the group. The group size was varied from 1 to 1 000, changing the number of calls needed to set the same number of elements according to the group size. For example, to set 5 000 elements one can call `dpSet` 5 000 times (once for each element) (2.76s), 2 500 times with groups of 2 elements (1.65s), 50 times with groups of 100 elements (0.5s) or 5 times with groups of 1 000 elements (0.7s).

By comparing the slopes of the different groupings, the optimal size for the groups can be estimated to be around 100 elements per `dpSet` (see Fig. 4.15). Figure 4.16 shows the dependence of the elapsed time on the group size. It is evident that when the group size is larger than 200, the elapsed time slightly increases. The optimal size of 100 elements is also working well for the smaller values as shown in figure 4.17. The optimal group size of 100 elements, based on the results of this analysis, greatly improves system performance. A normal implementation would have grouped as many elements as possible in a single `dpSet` request, leading to a very bad performance when a large number of elements has to be set.

# DPEs	Groups of 80	Groups of 100	Groups of 200
50	0.01	0.01	0.01
100	0.01	0.01	0.01
150	0.02	0.02	0.02
500	0.06	0.05	0.05
1000	0.1	0.1	0.09
5000	0.46	0.5	0.48
10000	0.91	0.86	0.97
15000	1.41	1.4	1.45
20000	1.88	1.88	1.93
30000	2.85	2.8	2.86
40000	3.77	3.79	3.88

Table 4.3: Summary table for the time elapsed for setting an increasing number of data point elements

Table 4.3 is a summary of the elapsed times for three grouping strategies around the optimum of 100. With groups of 100 elements, a linear

behavior with a slope of about 0.95 ms / 1000 elements is achieved.

Based on the conclusions of this study, a general function for setting the data point elements that automatically groups them in sets of 100 elements is widely used in the implementation of the control system.

4.11 Caching of Static Data

Along with dynamic data that changes in response to events coming from the front end devices, a control system must also handle a good amount of static information. Typical examples of static data are the structure of the hierarchical tree or the relations between data points representing different hardware parts. Handling this kind of data always implies a trade-off between flexibility and performance. The standard handling of the FSM tree in the JCOP framework favors flexibility (once the tree is modified, the changes are immediately reflected in the retrieval functions) at the cost of poor performance. The framework function for getting the children of one node (the basic functionality to perform a tree-traversal) is implemented as a `dpGet` that involves a significant overhead, as discussed in Section 4.10. Moreover, this approach tends to unnecessarily read data points that are not changing during normal operation.

The design of the Tracker Control System follows the opposite approach. The structure of the hierarchical control tree is cached thanks to the functionalities offered by a general library, named *treeCache*. At the time of configuration, all FSM hierarchy information (names and types of nodes, parent-child relations, data points corresponding to devices, etc.) is read once and dumped into a single data point. Each client using the *treeCache* library, either from a GUI or a script, reads the information from the single data point once and then keeps it in its local memory. In this way all the queries on the hierarchical tree structure are performed in memory with an impressive positive impact on the execution time. As an example, consider the problem of getting the names of all the power groups below the TEC plus node in the FSM hierarchy of the tracker. This query requires to visit 785 nodes, checking their type and selecting 384 out of them based on the type information. Using the framework FSM functions, this query takes around 40 seconds, while with the cache approach it takes around 0.6 seconds to read the cache from the data point (a procedure called only once when the manager starts) and 0.1 seconds to execute the query in memory.

The drawback of this choice is that in case the tree is modified, the cache data point must be updated by reading the complete structure again and all the PVSS managers using the cache must be restarted in

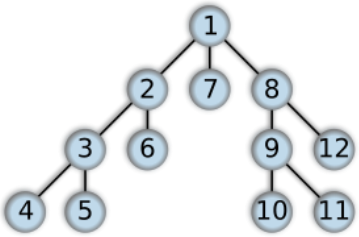

Figure 4.18: Order of visiting the nodes in a depth-first traversal of a tree. This order is the natural one for presenting a list of nodes to the user in any interface.

order to force them to read the updated information. This is not an issue for a production system, where the tree structure should change only in case of drastic expert interventions, such as the recabling of a power supply device.

To create the cache, a depth-first traversal of the tree is performed, assigning a sequential index to each node (see Fig. 4.18). Once the list of nodes is built and loaded into memory, in order to walk a subtree in depth-first order, it is sufficient to loop over all integers from the root's index to the index of its rightmost descendant. This order is also the natural one for listing the nodes in any user interface. The map is represented in memory as an array that lists all the node names and as a matrix that contains 19 items for each node (e.g., the PVSS system, the type, the parent, the children). The amount of memory used to store even a large tree is very low for today's standards (around 1 MB for the entire structure of the CMS tracker FSM tree).

In principle a general FSM hierarchy can contain references to the same node from more than one parent. If this kind of link is allowed, the hierarchy can be a graph rather than a tree (the parent of one node is not unique). In this case the common subtree is replicated by assigning different indexes to the same node.

4.12 Implementation of Protection Actions

In case of problems in the environmental parameters or in response to external events, the DCS intervenes by cutting off the power supply to a certain region of the detector. In this case, the normal switch-off sequence should not be applied, rather the commands should be issued to all the power supplies simultaneously. Moreover, in case of critical conditions the included state of the FSM objects should not be taken into account, because even a partition that is operated in stand-alone mode should

be switched off in case of emergency. For these reasons standard FSM commands cannot be used for executing protection actions.

The protection mechanism must also ensure that the portion of the detector concerned cannot exit the safe state as long as the critical condition holds. Any dangerous command should be inhibited at the lowest software level possible.

The protection actions are implemented in a general component, called *Detector Protection*, designed to meet the requirements of all CMS sub-detectors.

An action matrix, configurable for each PVSS system, defines a list of output data point elements to be set to a given value in response to each critical condition (e.g., the list of elements to be set for switching off the channels related to a portion of the detector). The locking feature provided by PVSS inhibits any further command on the output elements. When a DPE is locked by the protection manager, no other process can write to it. The output elements can be defined by specifying a pattern for the data point name or the alias, or by using the information stored in *treeCache*.

The *Detector Protection* component includes a mechanism for verifying that the protection action is executed succesfully. For example, the readback must be checked in order to ensure that all the affected channels are off. If the verification procedure fails, the elements are set again, to ensure that the safe state is eventually reached (e.g., if some commands are lost in the communication with the hardware). When the verification procedure succeeds, an acknowledgement is sent to the PVSS system generating the condition.

The tool is used for implementing a handshake protocol with the LHC. Before injecting the beam, the LHC sends a warning to all the experiments. In response to this signal, the protection mechanism brings all CMS sub-detectors to the safe state. When all the PVSS systems that have subscribed to the condition acknowledge that the safe state is reached, a confirmation is sent to the accelerator. The accelerator activities can then proceed.

The *Detector Protection* component provides a user interface to analyze the fired conditions and the currently locked data point elements.

4.13 Propagation Algorithm

4.13.1 Summarizing the State of the System

The problem of providing an effective summary for the conditions of a system including a large number of parameters is not trivial. The

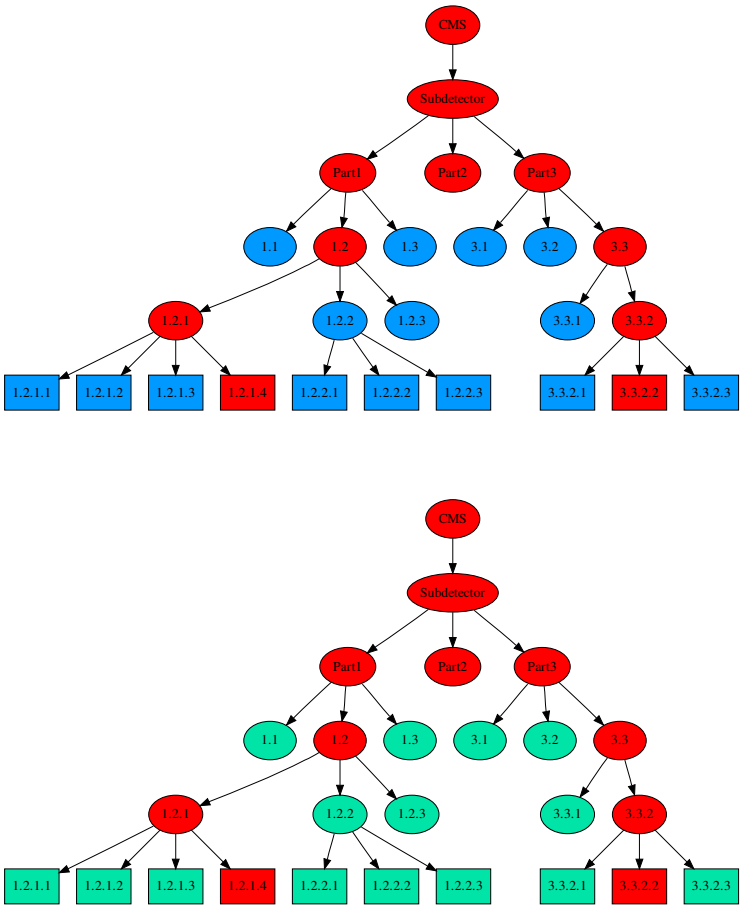

Figure 4.19: Example of loss of the operational mode in case of error (red nodes): it is impossible to determine (without browsing the hierarchy) whether most nodes are ON (green) or OFF (blue) in case some errors occur.

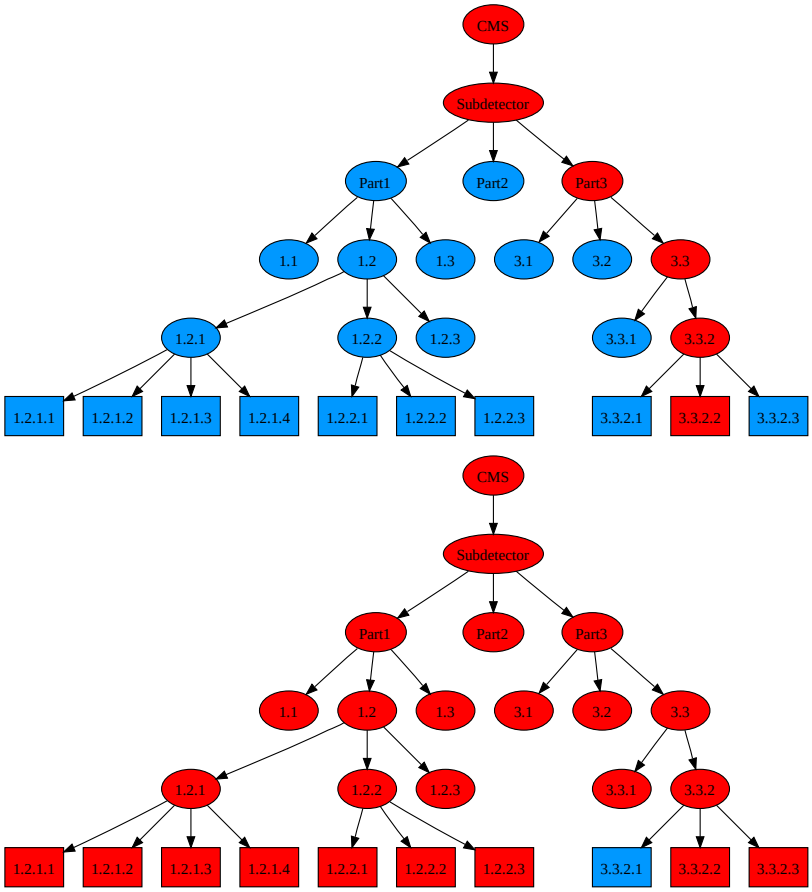

Figure 4.20: Example of loss of the quantitative information: the state of the top node is the same if one channel is in ERROR (red) or if almost all channels are in ERROR.

hierarchical structure is advantageous because, by partitioning in sub-components, it provides many abstraction layers and gives the user the possibility to act at different levels of detail. However, the choice to summarize the conditions of each node in a single state leads to two main problems:

Operational mode vs. Fault detection A control system must be able to cope with two types of information: the "operational mode", that is,. the overall status of the hardware (for example, OFF, ON, MIXED), and the "fault status", that is, the presence of errors. The standard approach, which is to combine these two aspects in a single state, can lead to the loss of information about the operational state when an error occurs (see Fig. 4.19). A possible solution to this problem is to provide two independent states reflecting both aspects in each node. That is the implementation choice in the ATLAS control system [35].

Loss of the quantitative information A second type of problem, not addressed by the double-state approach, is the loss of quantitative information in summarizing the state of a node. In fact, if the state is the only piece of information which is propagated in the hierarchy, it is not possible to weigh the nodes according to the number of devices they are related to. Moreover, with this classic approach, it is impossible, for example, to distinguish between the case where 1% of the devices are in ERROR and the case where 95% of the devices are in ERROR (see Fig. 4.20). A single channel that goes OFF could take the system out of the "ready for operation" state. Instead, it would be desirable to introduce some tolerance. Since the DCS state is typically synchronized with DAQ to notify when the system is ready for data taking, it is important not to stop the experiment in case of small singular errors.

On the other hand, the summary in a single symbolic state has some relevant advantages. A single state provides a useful level of abstraction which helps to reduce the complexity of the logic of the upper levels, minimizes the communication due to state changes and favors the independence of the nodes.

The original approach followed in the implementation of the tracker control system, presented in the next paragraphs, solves the two problems listed above by relaxing the property that the state of each node in the hierarchical tree depends only on the symbolic state of its direct children. Instead, the state of one node can be regarded as a function of the number of devices of a certain type in a given state included in the subtree related

to the node. This way we lose part of the modularity of the design but we gain a lot in the ability to effectively represent the state of a large number of devices. This approach is valuable in the case of large but homogenous systems, such as the tracker. The standard state-deduction approach is more suitable for high level controllers which must collect information from many sub-systems handling different types of equipment. The two strategies can easily be combined, by using the information computed with the new approach to integrate the standard FSM logic. This general approach was first presented in [33].

4.13.2 Assumptions on the Structure of the Hierarchy

Let us assume that a homogeneous system is built by devices of different types (e.g., LV channels, HV channels, temperature sensors), constituting the leaves of a hierarchical structure. The hierarchy is assumed to be a tree because this way the number of devices in each subtree is univocally defined. Each device type is typically characterized by some boolean properties (e.g., off/on, not in error/in error, ok/out-of-limit) that in principle can be completely independent (e.g., a channel could be in error no matter if it is on or off). Given n device types, each one with m_i basic states ($i = 1 \dots n$), the state of an internal node is described by $n + \sum_{i=1}^{n} m_i$ parameters, that are the number of devices in each state plus the total number of devices of each type in the subtree. From these numbers it is then easy to compute other parameters, such as percentages or summary states expressed in words.

The state representation of the system can be regarded as an abstraction procedure where a large number of parameters characterizing the system are "compressed" to fewer parameters that can still effectively describe the system. Assuming that the basic states of the devices fully describe the state of the system, a hierarchy with n device types, instantiated c_i times and each one described by m_i parameters is univocally described by $\sum_{i=1}^{n} m_i c_i$ boolean parameters. In the case of the tracker the algorithm handles three device types (one with two parameters and two with three). Hence a total of $356 * 2 + 3888 * 3 + 3888 * 3 = 25464$ boolean parameters are compressed in the top node into $n + \sum_{i=1}^{n} m_i = 3 + 2 + 3 + 3 = 11$ integer parameters.

Homogeneity is the criterion for deciding at what stage in the control hierarchy the use of the collected information should be replaced by a logic that combines the states of the children. In one detector which has different equipment for LV and HV power supplies, for example, the top node should combine the two states computed with the help of the

collected information for the two different device types.

4.13.3 Strategy for the Propagation of the Information in a Tree Structure

This section presents a general methodology for updating in real time information for all the nodes of a hierarchical tree. The algorithm is suited to any function of the state of the devices in which the value in one node depends only on the value of the same function in the children. Possible examples are: counting the devices in a state or sum, maximum, minimum value of a parameter. Although the mean does not satisfy this basic property (because the different subtrees must be weighted with the number of devices they contain), it can be computed with the same strategy, by propagating the sum and the total number of devices.

The strategy assumes that the leaves of the tree can arbitrarily change their state; the goal of the algorithm is to update, albeit with a certain delay, the values of the function at all the levels, while avoiding overloading the system. A naive implementation where any event causes the recomputing of the function in all the ancestors would lead to system overload, because the same information would be accessed and updated in parallel by many concurrent threads.

The idea of the algorithm is to collect the changes of the children in a certain time range before propagating the change to the parent in the hierarchy. To ensure that the topmost levels are eventually updated in case of continuous fluctuations, the state is propagated up after a certain number of events, regardless if other events are still coming from the lower levels.

For each internal node in the hierarchy, the algorithm uses a variable keeping track of the last thread which updated the node ($last\_updater$) and a counter of the rejected threads (num_kicked_out). The algorithm works bottom up. It is initiated by the state changing of one leaf ($child$). The event on a leaf starts a new thread executing the following steps:

1. Let $parent$ be the parent node of $child$ in the hierarchy
2. Set $last\_updater(parent) = my\_thread\_id$
3. Wait for a timeout
4. After the timeout elapsed, if still $last\_updater(parent) = my\_thread\_id$ (no other child updated the node) or if the number of rejected threads has reached the threshold ($num\_kicked\_out(parent) > max\_kicked\_out$) then:

- update the function for *parent*
- Set $num\_kicked\_out(parent) = 0$
- If *parent* is not the root, restart the algorithm with *child* = *parent*

else

- $num\_kicked\_out(parent)$=$num\_kicked\_out(parent)$+1
- Stop the thread

All the updates must be executed in mutual exclusion to avoid concurrency problems. Point 1 requires that the parent of each node is univocally defined, so the hierarchy must be a tree.

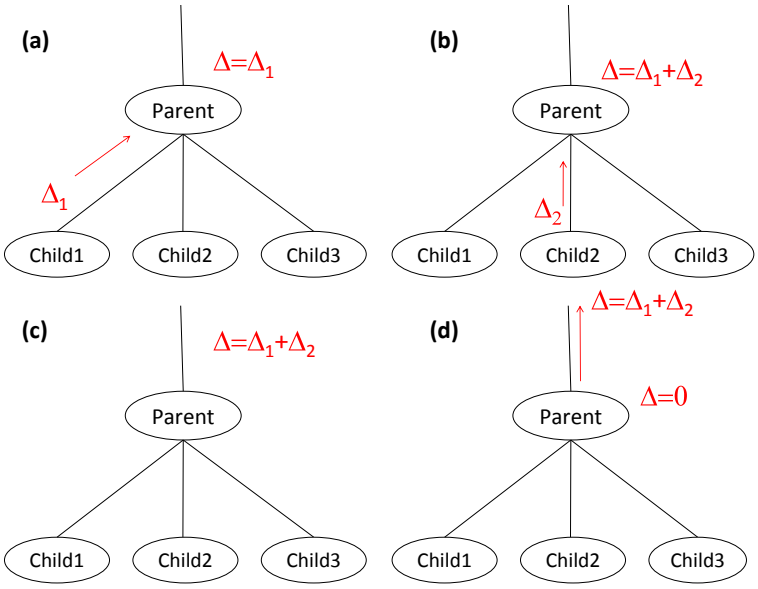

Figure 4.21: Propagation example for one node with three children: a first thread reports its change to the parent and then waits for a timeout (a). During the timeout, a second thread updates the Δ in the parent (and also waits for a timeout). The timeout of the first thread expires but the value in the parent has been updated by another thread so the first thread dies (b). During the timeout of the second thread, the value parent is not updated (c). When the timeout in the second thread expires, it finds that the value of Δ in the parent is the same it has written, so the second thread propagates up the change and resets the value of Δ in the parent (d). This strategy results in a great reduction of the number of concurrent threads.

When the function to be computed is just a counter (as in the algorithm to propagate the states), it is possible to avoid reading the values in the children that did not change and propagate the difference from the previous state (see Fig. 4.21), obtaining this modified algorithm:

- Before waiting for the timeout, a variable which collects the differences of all the children is updated: (*delta(parent)* = *delta(parent)* + *delta(child)*)
- When, after the timeout, the condition is satisfied, the number of devices is updated and the delta is reset to zero (*n(parent)* = *n(parent)*+ *delta(parent); delta(parent)*=*0*), then the difference is propagated up.

For this modified version, the condition *last_updater(parent)* = *my_thread_id* can be replaced with the condition that *delta*(*parent*) has not changed during the waiting time.

4.13.4 Implementation in PVSS

The algorithm for the propagation of the information, implemented in a library called *majority*, is based on the hierarchy tree defined by the JCOP FSM. In fact the algorithm was designed to extend and integrate the functionality of existing and well-tested control hierarchies built with this general framework tool.

The *majority* library works on the vectors describing the state of one node regardless of their meaning. In this way it is possible to provide a generic tool which handles the propagation of the information in the tree while a customizable user function computes the state of the devices. This approach is completely analogous to the implementation of the JCOP FSM (see Sec. 3.4.3), where the user has to write the custom functions to compute the state of a device unit of a particular device type. The configuration of the tool involves the definition of the device types, their possible states and the corresponding type in the FSM. Another user function is called to compute for each internal node an overall symbolic status depending on the percentages of devices in each basic state. The hierarchical structure must be a tree in order to define a unique counting for the number of devices. Therefore, multiple links are not allowed in the subtree handled by the library.

The modified algorithm presented in 4.13.3 runs in a control script. To navigate in the hierarchy, the *majority* library makes use of *treeCache*. The caching mechanism was a key point for ensuring a good performance, since the propagation algorithm needs to efficiently query the structure of the hierarchy. The script updates a data point for each node, containing the absolute numbers as well as the percentages and the summary state. This is the information that needs to be displayed in the GUI. The auxiliary information (*deltas* and counters for kicked out nodes) is only kept in memory.

To ensure a satisfactory response time to changes, the timeout must be kept quite short, especially when handling hierarchies with many levels. To avoid setting the data points too frequently, the propagation algorithm works in memory while a parallel thread copies the changes from the memory image to the data points, with a predefined refresh frequency. In this way a lower limit on the refresh time is imposed by construction and the script is not overloaded in case of massive state changes. However, in case of a stable state, the workload of the script is still low, since the script just checks the updating of the state values in memory.

The hierarchy can be distributed in several PVSS systems: in this case the *majority* CTRL script must run in all the different systems and the propagation of the information from one system to the other is handled by connecting to the lower level data points from the upper system. This is needed because the scripts running in two different PVSS system do not share the memory and must communicate via a data point connection.

The integration of the states computed by the *majority* library in the FSM is obtained by adding the data point related to each internal node as a child of the related FSM node. Additional logic can be programmed in the FSM to set the state of the node taking into account the additional information computed by the *majority* script. Instead, if no change in the FSM state is needed depending on the quantitative information, the user can choose to use the standard FSM logic to compute the state of the nodes. In this case, the information provided by the propagation algorithm can just be displayed in the DCS GUI for integrating the information provided in the FSM state.

A key requirement for the integration of the propagation algorithm with the JCOP FSM is to take into account the inclusion states of the nodes in the computed numbers. When a node is excluded from the FSM hierarchy, all the devices in the subtree below it should be excluded from the total number, so that an excluded node does not influence the status of the parent. Handling of inclusion and exclusion of nodes is possible using the same algorithm as for the propagation of the differences by acting on the total number of devices. The script computes independently the counting of the number of included devices and the total number regardless of the inclusion state.

The total number of devices in each subtree can of course be computed *a priori* from the static structure of the hierarchy. Nevertheless, the total number of devices is also computed by the script during startup, giving a good criterion for deciding when the script is completely initialized.

The package comes with some panels, that provide the elements to be

used in the GUI for displaying the percentages and the absolute number of devices in each state. The user interface is completely generic and reads the device types and their states from a specific configuration data point. The panels are customizable for showing only some of the devices/states, to adapt to the users' needs. The user interface also includes a basic search feature that implements a tree traversal to find the devices in a given state.

4.13.5 Customization for the Case of CMS Tracker

State of the Power Supplies

The CMS tracker uses the *majority* library to compute the state of the power supply system and manages three types of devices (control channels, LV channels, HV channels), each one with two basic states (off/on, not error/error). Additional information about the percentage of inhibited power groups (not ready to be turned on) is also computed. This feature is discussed in the next section.

Since the DAQ has to wait for the tracker to be in ON state to take data and stops when the detector exits from this state, it is important to introduce some tolerance to keep on running even if some modules are not powered. For this purpose, the FSM should move from "mixed" to "pure" states when the percentage goes above a certain threshold (typically 95%). For the same reason, an error in a single power supply channel (e.g., a trip) should not bring the entire tracker in ERROR state, so the transition to the ERROR state is ruled by another threshold (typically 5%). The states for the nodes above the control group are listed in Table 4.4. The "mixed" states are needed for safety reasons, because they report that at least one channel is ON. For example, a single switched-on HV channel is enough to bring the entire detector into the HVMIXED state. This state signals that the detector is not ready for beam injection.

The *majority* scripts are running on five PVSS systems: the four controlling the mainframes and the supervisor. Since the state of the LV and HV channels are both related to a power group, a device unit in the CMS tracker FSM, the initial counting of the number of devices in each basic state is performed by the script handling the state of power groups (see Sec. 4.3). This script provides, in addition to the state, a bitmask describing the state of each of the four channels.

The percentages computed in the *majority* script are presented to the user with the help of a color convention following the framework guidelines (see Fig. 4.22). A new color that provides feedback when 100% of the devices are on (see Fig. 4.23) is introduced.

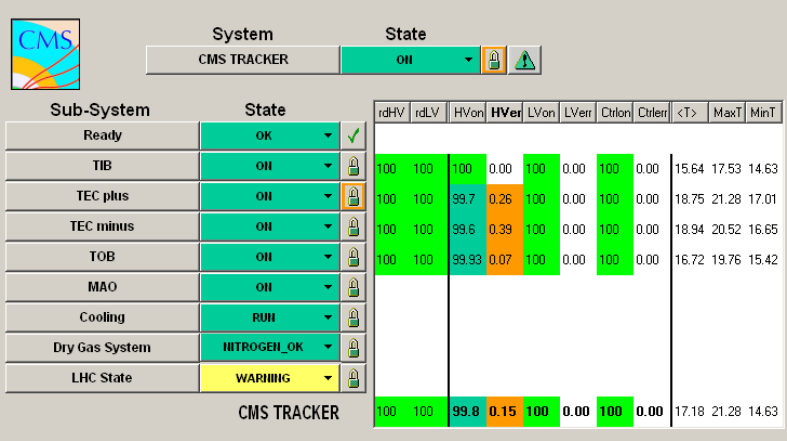

Figure 4.22: Summary table for the percentages of channels in the basic states for the CMS tracker. The *majority* library keeps track of the state of three device types: control channels, low voltage channels and high voltage channels. For each device type, the percentage of channels ON and channels in ERROR is counted. In addition, the number of LV and HV channels ready to be switched on is computed, in order to give to the user a useful feedback of the inhibit conditions. The mean, maximum and minimum temperatures in each node are displayed beside the percentages.

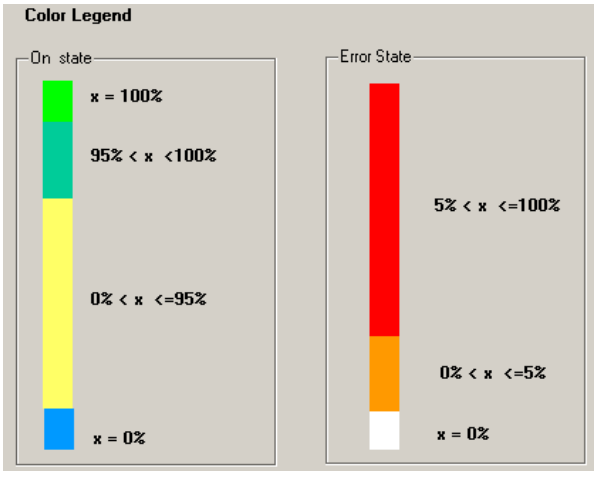

Figure 4.23: Legend for the color convention used for displaying the percentages. The convention uses the standard framework color but a different kind of green is used to give the feedback that all the devices are in the final state.

State	Ctrl On	LV On	HV On	Err.
OFF	0	0	0	$x \leq 5\%$
CTRLMIXED	$0 < x \leq 95\%$	0	0	$x \leq 5\%$
ON_CTRL	$x > 95\%$	0	0	$x \leq 5\%$
LVMIXED		$0 < x \leq 95\%$	0	$x \leq 5\%$
ON_LV		$x > 95\%$	0	$x \leq 5\%$
HVMIXED			$0 < x \leq 95\%$	$x \leq 5\%$
ON			$x > 95\%$	$x \leq 5\%$
ERROR				$x > 5\%$

Table 4.4: Meaning of the states in the upper nodes of the Tracker hierarchy. All the (non empty) conditions in one row must be valid in that state.

The online computing of the percentages in the tracker is an efficient way to search the devices in a given state by walking the tree and following the branches where some devices are in the state being searched.

Readiness of the Power Supplies

A guiding principle in the implementation of the Tracker Control System is to keep the FSM hierarchy as simple as possible and to only reflect the status of the power supply system. For this reason, various conditions coming from the environmental monitoring or from external systems are not directly taken into account in the FSM status, but only inhibit the switching on of the relevant power supplies. However, following this strategy the user may give a high-level command and the inhibit conditions prevent the switching on of a substantial part of the tracker without giving any direct feedback.

In order to introduce a useful feedback, the counting of the number of "not-ready" LV and HV channels was introduced in the online computing of the states.

There are several conditions that can inhibit the switching on of LV and HV, acting at different levels of detail. Among them:

- some global conditions, such as the general cooling plant status, act on the entire tracker
- the environmental readings or the cooling status (when the software threshold on the temperature is reached or if the cooling loop valves are closed) act at the level of sector / cooling loop

- an error on the control group channel inhibits all the power groups in the control group

Moreover, there are some external global conditions, such as the LHC state, that should only inhibit the HV and not the LV.

The protection actions are handled in the *Detector Protection* component (see Sec. 4.12), that, in response to a critical condition, switches off the affected channels and prevents them from being switched on again with the locking mechanism. The locked state of the channel is propagated in the hierarchy using the propagation algorithm.

In this way the operator can see the percentage of not ready devices and, if some part of the detector is inhibited and cannot be switched on, he can access a detailed panel and investigate the cause of the inhibit.

This feature was introduced in a second iteration to the functionalities of the CMS tracker majority. However, the changes needed were very limited and only affected the user functions, proving the flexibility of the *majority* library.

On-line Computing of Mean, Minimum and Maximum Temperatures

A variant of the propagation algorithm is used for computing online the minimum, maximum and average temperature of the top nodes in the hierarchy. In this case the algorithm is implemented directly in the script computing the state of the temperatures at the cooling loop level (see Sec. 4.5) and does not use the *majority* library. Since in this case the algorithm has to deal with floating point numbers, a smoothing algorithm is introduced to avoid the propagation of insignificant changes. The smoothing mechanism reduces the number of events to be handled, allowing an efficient operation of the algorithm. The first smoothing is done at driver level by ignoring small fluctuations in the reading of the PLC sensors. Thanks to the smoothing mechanism, the events are not synchronous with every polling cycle from the PLC. However, the changes in temperature of the different sensors are not independent. When a part of the detector is switched on or off, temperatures are consistently increasing or decreasing causing a peak of events. The propagation algorithm is able to treat these peaks properly by discarding most of the events and recomputing the relevant parameters after the configured timeout.

The propagation of these values immediately allows the user to identify very high or very low temperatures by using the information collected in the hierarchy to increase the efficiency of the search (see Fig. 4.22).

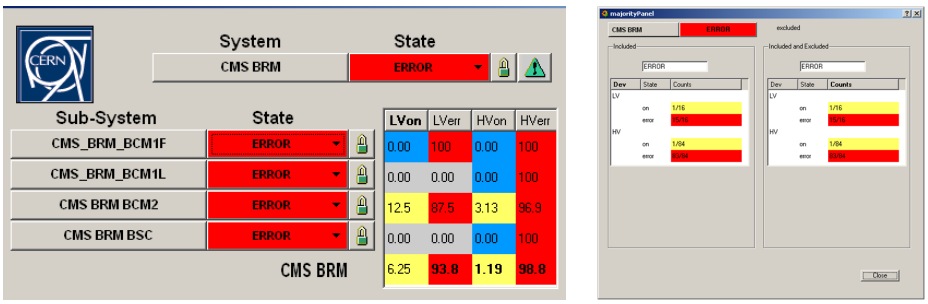

Figure 4.24: Summary table with percentages and detailed table with absolute numbers reporting the counts of the propagation algorithm in the BRM power supply system

4.13.6 Application of the Propagation Algorithm to the DCS of Other Sub-detectors

Beam & Radiation Monitoring Power Supplies

The Beam Radiation & Monitoring for CMS (see Sec. 2.4) is a complex safety system used to monitor beam quality. It can trigger a beam abort in case of unsafe conditions that causes the deflection of the beams into the beam dump and shuts down the "at-risk" detectors.

From the point of view of the control, the power supply system must be integrated in the control of the experiment. Despite the relatively small number of devices compared to the tracker (a total of 16 LV and 84 HV channels) a tool counting the number of channels in ON or in ERROR can still be useful. In this case, the customization of the package is straightforward since both types of channels are handled as a device unit in the FSM hierarchy.

The test of the *majority* package in the BRM system showed that the strategy, though designed for large systems, is also suited to smaller systems. From the point of view of the package user, the programming work turned out to be very limited. It is sufficient to define the types of devices that must be handled and the behavior of each device type. The callback function to compute the basic state of a device could be easily adapted from the definition of the DU state in the JCOP FSM or directly connect to the FSM state. The library comes with some sample code for these procedures that could be easily customized. Comparing to the definition of the FSM logic, the customization of *majority* is easier and faster because the user does not need to program any kind of specific logic for the high level nodes. Also, the user interface is immediately available thanks to the generic panels included in the *majority* package (see Fig. 4.24).

CMS Drift Tubes

The DCS for the CMS Drift Tubes (DT), that are part of the CMS Muon System, is also using the *majority* library for determining the state of the power supply system.

The hierarchy for the DT reflects the detector geometry. The DT system is divided into 5 independent wheels. Each wheel is geometrically segmented into 12 sectors. Each sector is equipped with 4 chambers, except the horizontal top and bottom with 5 chambers each. Each chamber is divided into 8 or 12 layers. A layer is a complex object including 4 HV channel and a so-called "macro-channel", but it is treated as a DU in the DCS. As for the power groups in the tracker, a separate script computes the status of the layers independently from the FSM. Additionally, each chamber needs 4 LV channels. Hence the control system has to deal with a total of 60 sectors, 250 chambers, 2720 layers and 1000 LV channels. From the point of view of the DCS, the size is therefore similar to the tracker. The size and the homogeneity of the DT DCS qualify it as a proper use-case for the propagation algorithm.

The *majority* library is used in the DTs to count the number of layers and LV channels in the different basic states. For the LV channels, the relevant properties are off/on and not error/error. For the layers, three different properties have to be counted: number of ON layers, number of' "partially ON" layers (corresponding to the ramping phase of the HV channels), number of layers in ERROR. So the state of an internal node is defined by 5 parameters.

The DT DCS is distributed over 6 PCs. One is used as a supervisor handling the upper layers of the FSM and all the LV channels. Five PCs (one per wheel) are devoted to HV control. The *majority* script runs on all the PCs. The standard user interfaces are used for displaying the percentages.

CMS ECAL Control System

The CMS ECAL Detector Control System [36] provides the monitoring of the detector conditions, of the on-detector electronics and of all ECAL subsystems (High Voltage, Low Voltage, Cooling System, status of laser monitoring system). ECAL DCS adopted *majority* to compute the number of LV and HV channels in the basic ON and ERROR states. ECAL DCS handles a total of 860 LV channels and 1240 HV channels, controlled by individual DUs.

The CMS ECAL control hierarchy is driven by ECAL subsystem structure and geometry. The top node is divided into 6 TTC partitions, further partitioned into "supermodules". Each supermodule han-

dles different subsystems, providing the different services (HV, LV, Cooling, Safety System, etc.). The use of *majority* applied to LV and HV systems, allows for the counting of the number of channels in ON and ERROR. This counting is used to introduce a certain error tolerance in the FSM states.

ECAL DCS is distributed over a total of 14 computers; *majority* runs on 9 of them (three for LV, five for HV and the supervisor).

4.14 Wizard for Error Diagnosis

A typical problem that must be addressed in a complex control system is the analysis of error conditions and the implementation of an effective recovery procedure. Depending on the type of problem, an automated recovery procedure may be possible or the intervention of an expert may be needed. In the standard approach, the analysis of the situation is usually left to an expert (or a highly trained shifter) who deeply understands the architecture of the system and all interconnections between the different hardware. Still, a human expert could spend a considerable amount of time to find all the information needed to identify the problem and to execute the recovery procedure manually. Moreover, when a problem occurs, the typical reaction of a shifter is to try to bring back the system as soon as possible into normal conditions in order to be able to restart the operation. To achieve this objective, he will try several procedures that might be able to solve different kinds of errors, until he finds the working one. This approach is harmful, because at the end of the procedure the user will not have a clear idea of the cause of the problem, will write confusing logs stating that at some point the error somehow cleared but will not be able to tell which procedure he followed in order to solve the problem.

To avoid this kind of interaction, the strategy of the tracker control system is to avoid relying on the competence of the shifter and to automate the recovery procedures as much as possible, in order to identify and automatically log the causes of the error.

As a matter of fact, in most cases a computer program can analyze the various sources of error in a much more effective way than any human expert. Such a wizard can also be used by untrained shifters who are provided with a friendly user interface (see Fig. 4.25) adopting an extremely simplified user interaction. This approach minimizes the downtime due to errors and relieves the experts from unnecessary work.

A knowledge-based system may seem to be the ideal implementation for this kind of wizard. In some applications, knowledge-based systems

Figure 4.25: The user interface for the CMS Tracker Wizard, called on the CMS tracker top node. The user interaction is limited to two or three possible choices. The expert can choose to perform only some of the different kinds of checks.

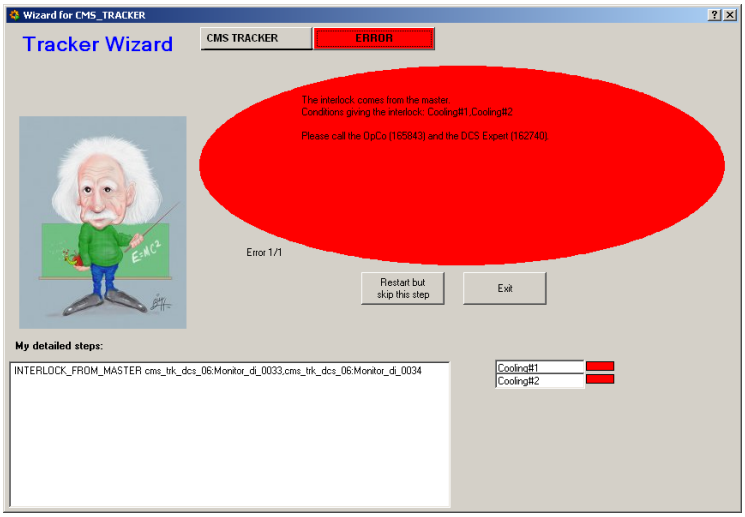

Figure 4.26: The CMS Tracker Wizard notifies the user of the cause of the interlock condition. In this case the wizard gives directly the phone numbers of the relevant expert.

can be integrated with a SCADA system to perform specialized tasks, such as Fault Diagnosis or Alarm Processing. However, the implementation of the wizard uses a hardcoded algorithm rather than a knowledge database. In fact in knowledge-based implementations, data from the SCADA system is often assumed to be immediately available to perform the analysis. Instead, the primary goal of the present implementation is to minimize the amount of information retrieved from the SCADA system in order to provide a recovery procedure with a good response time, by querying only the data needed for the current step. Moreover, the decision process is not particularly complex and is well documented.

The wizard works in two phases. In the first one, the status of the system is analyzed. If the error condition is not clearable, the user is notified with the cause of the error (for example, if a global condition that interlocks the entire tracker is still present), so that he can call the appropriate expert (see Fig. 4.26). If a recovery procedure is possible, the output of the analysis is a sequence of instructions that should be executed in order to bring the system back to a normal situation. In this case, the wizard proceeds with the second phase, that is, the execution of the instructions, giving the user a feedback of the current operation. Some steps in the procedure can fail (for example, if the hardware does not react as expected, after a certain timeout), and in this case the user is notified. When the procedure is successfully completed, the user is notified of the success. The wizard never commands the switching on of parts of the detector. Instead, when the recovery procedure is completed successfully, the system is taken to a non-error state (most likely a partially OFF state) and it is ready to receive the proper command from the user.

The analysis is divided into separate steps that take care of different types of errors (connection problems with the hardware and with the DCS PCs, problems in the 48V power converters, interlocks, non communicating boards). For each of these conditions a fast check indicates whether it is necessary to analyze the status of the hardware in more detail or if the detailed analysis can be skipped.

The wizard makes use of the caching mechanism for retrieving the structure of the hierarchy. To identify the parts of the detector in error, it reads the counting of the devices updated by the propagation algorithm. Information from the custom configuration database is used to obtain the cabling relationships. The wizard does not need a direct connection to the custom configuration database, because all the relevant information is stored in specific data points (see Sec. 4.7).

The following steps describe the analysis algorithm used by the wizard. The analysis can result in a success (the recovery is possible) or in

a failure (the recovery is not possible). The two possible outcomes are marked as SUCCESS and FAIL.

- The wizard is called on one node, that is the root of the tree to be analyzed.
- Find the minimum subtree in error under the given root and limit the analysis to this subtree. In this way the wizard avoids performing expensive analysis if the error is limited to a small number of nodes.
- Analysis of the status of communication with the hardware:
  - Check that communication with the controlled hardware (PLC, power supply mainframes) and with all the DCS PCs is working. In case of problems, report to the user (FAIL).
- Analysis of the status of the 48V service power:
  - Check the status of the 48V channels related to the given partition (plus side, minus side). If they are all ON, this step can be skipped.
  - Identify the racks or crates in ERROR as a result of missing 48V power.
  - Check the corresponding 48V channels. If one channel is OFF, switch it on (this action requires a user confirmation). If one channel is in ERROR, report the fact to the user and suggest the user to call the power supply expert (FAIL).
- Analysis of the interlock conditions:
  - Check if the Master PLC is interlocking as a result of a global condition (e.g., Cooling Plant Failure). In this case report the fact to the user (FAIL).
  - Check if the interlock is forced in the PLC by the user. If the interlock is forced, remove it (SUCCESS) or report the fact (FAIL) depending on the user role.
  - Find the interlocked crates.
  - Identify the relays connected to the interlocked crates.
  - If the connected relays are not fired, acknowledge the power supply system to remove the interlock condition (SUCCESS).

  - If there are some fired relays, find the interlock groups that can fire the relays and verify if the interlock condition still holds (check number of out-of-limit sensors in each group).
  - If the interlock condition still holds, report the fact to the user (FAIL).
  - If the interlock condition does not hold anymore:
    * Acknowledge in the PLC the sensors belonging to the group.
    * Wait for the relays to remove the interlock.
    * Acknowledge the power supply system.
    * Acknowledge the alerts in DCS (SUCCESS).

- Analysis of the non communicating boards:
  - Find all the non communicating boards in the subtree.
  - Check if they all belong to a same crate.
  - Report the results (FAIL).

- Analysis of inhibit conditions (see page 111):
  - Find inhibited subtrees.
  - Report the causing condition to the user (FAIL).

- Analysis of power supply conditions:
  - Find channels in ERROR (trip, over current etc).
  - If some conditions are not clearable (because the problem still holds), report the fact to the user (FAIL), otherwise acknowledge the power supply system to clear the error and bring back the channel to OFF (SUCCESS).

In case of success, the wizard executes the instruction to solve the problem. If the sequence is executed successfully, the wizard checks if the node is still in error. In case some error is still present, the analysis is restarted. This strategy allows solving different kinds of problems in several steps.

The wizard can be used from an interactive interface but is also automatically called without user interaction. This feature is used when the tracker is operated centrally by the CMS control system.

The wizard works on the standard FSM hierarchy for the control of the detector and only analyzes the state of the hardware related to the

subtree on which it is called. This allows the user to recover the state of one of the main partitions regardless of if the other partitions still have a problem.

4.15 Definition of Alerts

Alerts and FSM error states are two ways to identify the faults in a large control system. Though the two methods can have some intersection, they are mostly complementary. The ERROR state of one device usually corresponds to a hardware status, such as a fired safety feature that must be acknowledged before bringing the system back to operation, or an interlocked status of a power supply. On the other hand, alerts indicate that a certain parameter is outside the expected range and are used to attract the users' attention on a particular device. Alerts can be defined on binary values, when one of the two states corresponds to an error condition. For example, the Trip or the Over Current state of a power supply channel generates an alert. For floating point values, alerts can be used to signal that the parameter is outside of the normal operation interval. Alerts are typically defined for monitored currents, voltages or temperatures. In the case of floating point values, several alert ranges can be defined for the same element, giving rise to alerts of different severity.

The number of alerts should never explode in case of overspread problems. In general the alerts should be defined in a way that a single source of error corresponds only to one alert and not to many. For example, if a 48V power converter channel is switched off, an entire crate goes to an ERROR state because the power supplies do not receive the needed service power. In this case it is preferable to define an alert on the cause (the OFF status of the 48V converter channel) rather than on the effect (the "48V is missing" error of the power supply boards). For the same reason, in case of an interlock, the alert is defined on the status of the PLC relay (the causing condition) rather than on the interlocked bit of the power supply board (the effect).

The alerts on the temperature values are not defined on the single probe reading but rather on the mean, maximum and minimum value defined at the level of cooling loop and sector. This strategy reduces the number of alerts in the system while still providing the user with an effective tool to identify the region of the detector where the problem is located.

An alert can be programmed to execute an action corresponding to its state transitions. These actions can be used to play an audio alarm

on the user interface, to notify the experts via SMS, or to keep track of the error conditions in a specific log.

A particular way of using alerts is to define safety limits on the read back settings of certain parameters. For example, the power supply safety limits on the monitored current or voltage can be programmed over a large range. Even if the experts may need to adjust the values, there are some limits that should never be exceeded in order to ensure detector safety. Of course the control system does not allow any user to set the parameters outside the safe region, however, the limits can be exceeded when a power supply module is replaced with another one that is not properly configured. The safety limits are configured in specific alerts defined on the read back settings that can set the corresponding power group or channel to the not ready status (see Sec. 4.13.5) and inhibit the powering of the channel.

4.16 Periodical Checks

Alerts are an effective tool for spotting problems in the parameters depending on their values. However, there are some anomalous conditions that can only be identified by analyzing the historical trend of the parameter or by comparing it to other related readings. These problems do not normally require immediate intervention but must be reported to the experts or to the shifters and logged for further investigation.

Another kind of problems that must be identified are software and network failures, relatively common in a distributed system. This kind of problems cannot be solved by the standard DCS shifter but require intervention of the DCS software expert. This class of problems is not sub-detector specific and can be addressed by a general tool. The CMS system overview tool [37] is designed to monitor the control PCs and provides this kind of functionality for the entire experiment. It will eventually replace the custom solution developed for the CMS tracker.

Both kinds of checks are implemented in control scripts that periodically query the system conditions and report to the experts any anomalous condition via mail or, in case of severe problems, by SMS. These scripts are running on a separate dedicated PC, connected to all the other PVSS systems.

For the first class of problems (cross-check of system parameters and trending analysis), the following conditions are detected:

- switched on power supply channels with zero or low current, corresponding to a loose cable

- power supply modules losing the communication to the branch controller
- power supply modules with wrong parameters, not matching the configuration database
- rising temperatures or dew points that can be detected before they hit a warning level.

while some examples of software and network checks are

- connection problems between the PVSS systems
- communication loss with the hardware
- scripts not running

The key strategy in the implementation of the periodical check is that all the procedures are based on massive `dpGet` rather than on `dpConnect`. In this way, the periodical checking procedures can work independently from the normal system operation and do not have any significative impact on the performance of the system since they do not subscribe to any parameter. The interval between two subsequent checks can be tuned depending on the severity of the critical condition and on the complexity of the query.

An automatic analysis of the monitored parameters relieves the users from routine work, attracting their attention on unusual trends or values. In many cases, the expert can be notified directly without shifter intervention. Once the problem has been reported to the expert, he can take protective measures before the parameters reach a critical level. When the number of parameters to be monitored is huge, the periodical checks can spot problems that would otherwise go unnoticed.

4.17 Graphical User Interface

4.17.1 Principles

The design of a Graphical User Interface (GUI) for a complex control system must provide a simple interface for non-expert shifters, while allowing the expert to act at a very detailed level in case of problems or for performing a specific analysis.

The amount of information to be displayed (especially to the non-expert users) must be accurately balanced. An information overload

could cause difficulties in the interface usability and preclude an immediate understanding of the status of the system. On the other hand, all the essential information should be displayed and tools must be provided for easy navigation of the hierarchy in order to reach any specific detector part in case of problems.

The user interface has to satisfy some constraints arising from the need to integrate the Tracker Control System in the overall CMS control system. The FSM interface is defined in the framework and provides a generic structure for the user panels, with a space that can be customized for each FSM object. The standard FSM interface (see Fig. 4.27) shows the state of the current node at the top of the page and, below, the state of its children in the hierarchy. Each node has a specific panel that can be opened to display its detailed state. Commands can be given by clicking on a state and a tree navigator on the left provides the access to the detailed panel of any node.

In the tracker GUI, the percentages computed by the propagation algorithm (see Sec. 4.13) are displayed beside the states, giving detailed information about each node. In this way the symbolic state shown in the standard interface is integrated with the quantitative information that gives the "state of the node" in terms of number of devices in given basic states (see Sec. 4.13.2).

The design of the interface for the FSM panels follows a criterion of uniformity. In order to facilitate usability, some basic elements are used in all the panels. Of course the user has to be familiar with the metaphor of the hierarchy used in the representation of the detector, because all user interaction is based on that. For example, the menus that give access to all specific functions always refer to all the devices below the node where the command is given. Since the parent-child relation in the hierarchy usually represents a physical container-contained relation, this metaphor is quite intuitive. The navigation in a hierarchy is a familiar task for computer users because it is found in many contexts, from filesystems to the web.

The framework recommendation for the shifters is to display the Alarm Screen (see Fig. 4.28), that is, an interface that displays all the current alerts. This panel integrates the information provided by the FSM by displaying the list of current problems. The Alarm Screen allows the shifter to easily reach the specific panel corresponding to the problematic device. The number of alerts must also be balanced to avoid the situation where a single problem is reflected in a large number of alerts (see Sec. 4.15).

Another kind of user interface that is often used in control systems is the trend over time of the parameters (see Fig. 4.29), directly connected

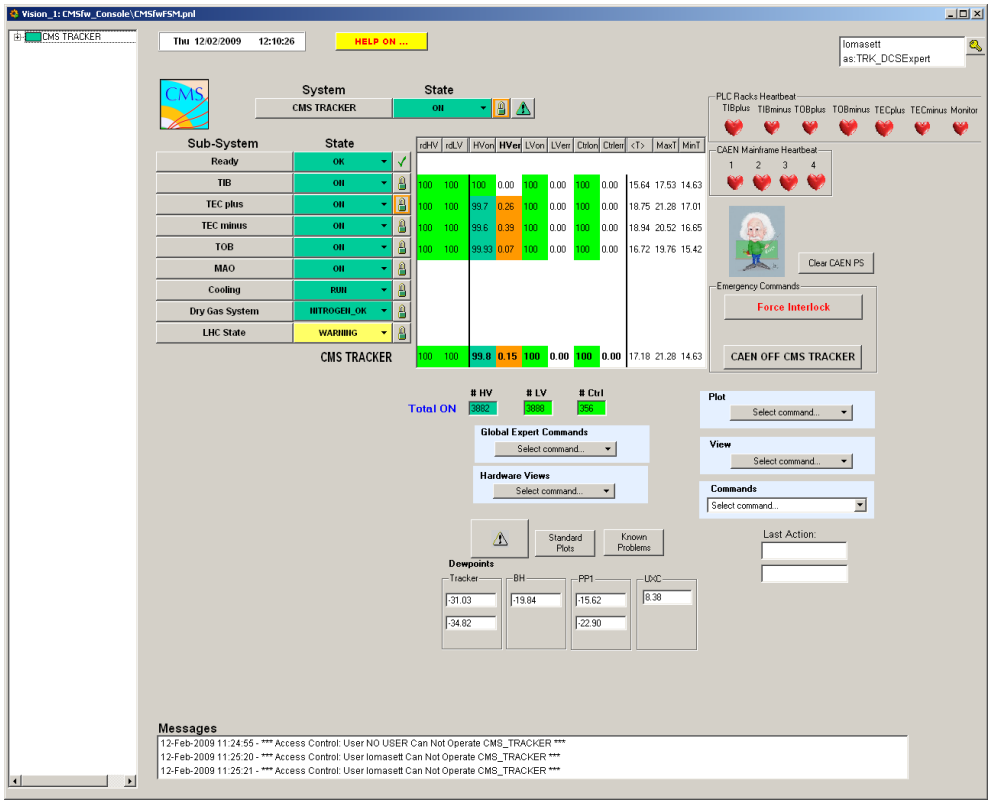

Figure 4.27: Screenshot of the main user interface for control of the CMS Tracker. The panel shows the state of the top node and of its children nodes. By clicking on one node, it is possible to give commands by choosing the command from a drop down list. A double click on one of the children opens a new panel with detailed information. The percentages of devices in each state (computed with the propagation algorithm) are displayed for each node beside its state. The same table displays the mean, maximum and minimum temperature for each partition. Thanks to the information summarized in the table, the shifter can immediately identify the error and click to open a detailed panel that allows him to find the specific nodes in error without need of browsing the hierarchy. On the right, some heartbeats give the feedback that the communication with the hardware is working. The wizard for the automatic diagnosis and recovery of errors can be opened by clicking on the icon of Einstein. The emergency commands (forcing the interlock in the PLCs or giving an emergency OFF to the entire detector) are accessible with two large buttons. A confirmation is required to avoid unintentional commands. Some drop down menus, divided in various categories give access to the plots and specialized views. The button with the danger sign opens the alarm screen.

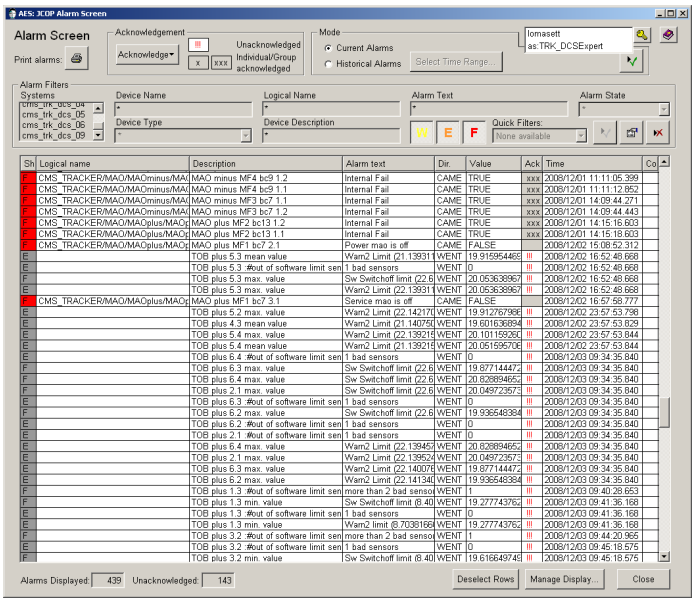

Figure 4.28: The alarm screen interface. All the currently active alerts are listed in the interface that includes information about the time of the last transition and the current values. By clicking on a specific row, the detailed panel of a certain device can be opened, allowing the user to investigate the problem.

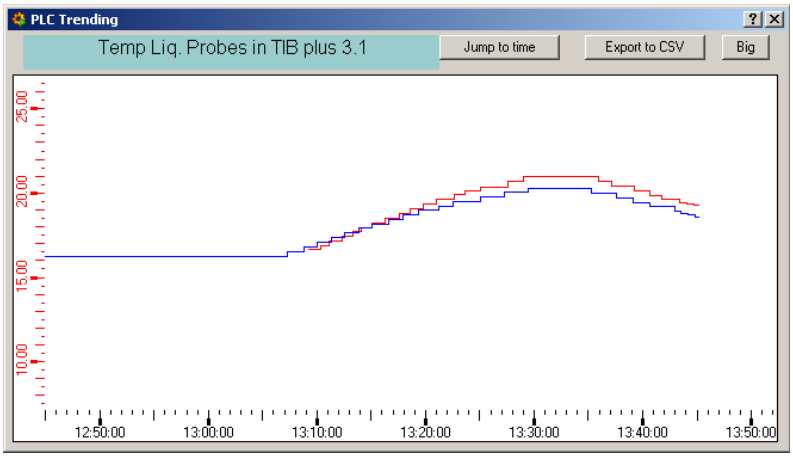

Figure 4.29: Example of the trend of the temperatures in one cooling loop of the TIB

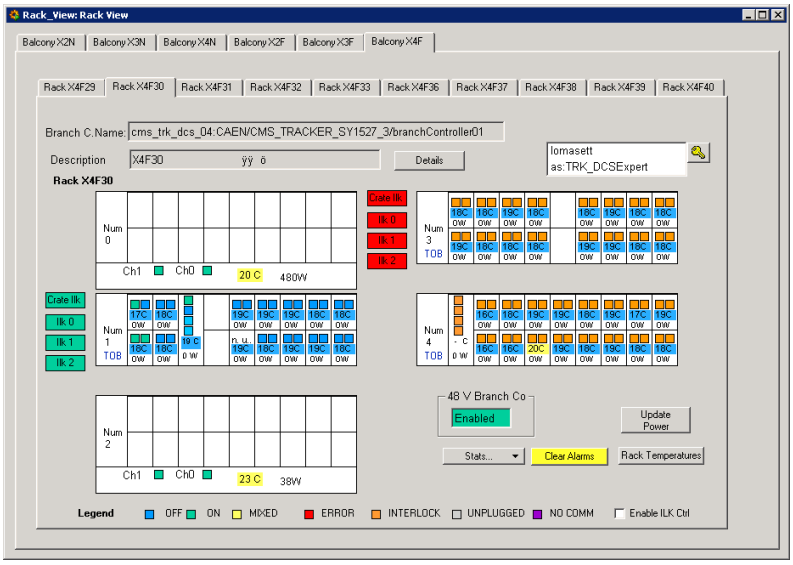

Figure 4.30: Screenshot of the rack view panel presenting the status of all the power supply units in one rack. The PLC relays that can cause an interlock are also presented in this panel.

to the conditions database (see Sec. 3.4.4). Trends facilitate the analysis of the evolution of the conditions and of the correlations between different parameters. A trend of a large number of parameters must be avoided because it leads to an overload of the system and to confusing plots.

The expert is provided with some panels that give a view of the auxiliary hardware (power supplies and PLC) relative to their position in the rack. For the power supply system, a panel called rack view (see Fig. 4.30) summarizes the state of the power supplies in one rack with a color convention. By clicking on one particular board, the expert panel for the power group (see Fig. 4.32) can be reached. An analogous panel shows the value of the PLC probes, arranged according to their connection to the PLC rack (see Fig. 4.31). The detailed panel of the specific devices provide a link between the logical and the hardware view, by reporting both the logical name and the hardware location.

Finally, an innovative kind of user interface for control systems is the three-dimensional GUI. There are several issues in developing this type of interfaces. A preliminary study investigating the possibility of using a formal model-based methodology in the development of a 3D GUI for the control of the CMS tracker is presented in chapter 5.

Figure 4.31: The rack view for the PLC probes. In this hardware view, the probes are displayed according to their connection to a specific row, slot and channel in the PLC rack. The status of the relays is also displayed.

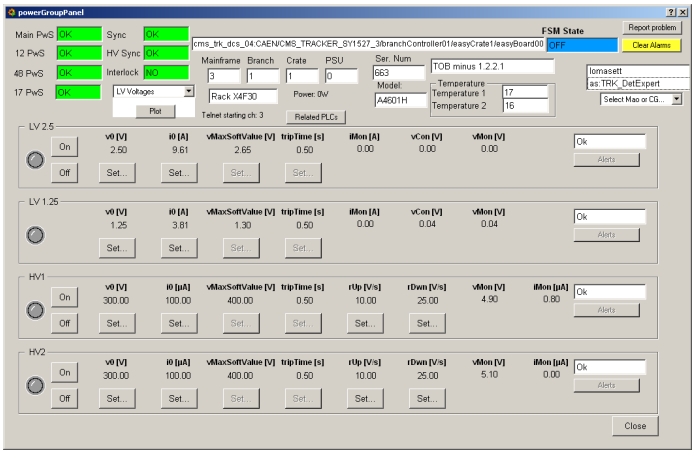

Figure 4.32: The expert panel for a power supply unit powering a power group (2 LV and 2 HV channels). The parameters are displayed and the settings can be changed by the experts.

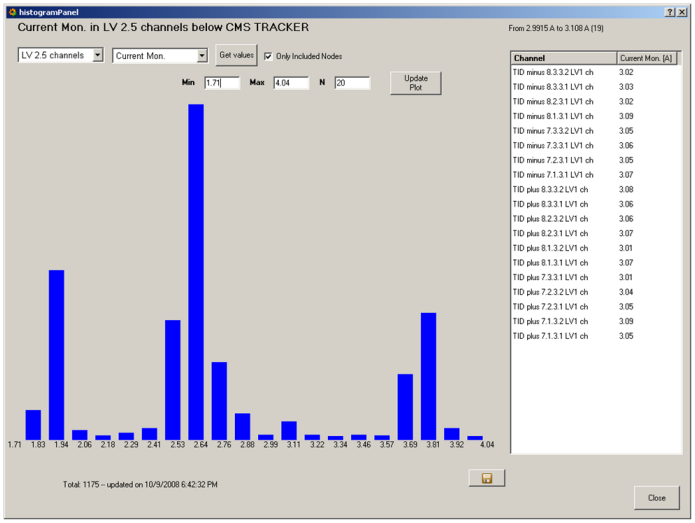

Figure 4.33: Example of the histogram GUI for displaying the distribution of the monitored current in all 2.5 V channels in the tracker (a total of 1175 included channels). The bins are clickable and open, on the right side. the list of devices whose values are included in the interval. Clicking on the individual channel then leads to the specific device.

4.17.2 Tools for Fast Information Retrieval

Typical tasks that the detector expert may want to execute in a control system for a HEP experiment require the analysis of the large number of parameters. When discussing with the users the features needed, a common request is to have a way of displaying the value of a certain parameter (e.g., the monitored current) in all the channels (or probes, or DCUs) of the system. Even if at a first glance it could seem reasonable, it is not possible to visualize a large number of parameters by just displaying their numeric values. Even with the help of coloring conventions, any attempt to display more than 20-30 parameters on a user interface prevents a human user from performing any useful analysis in a reasonable time. Since in the tracker the users have to deal in many cases with more than 1000 parameters[1], a more effective interaction strategy had to be designed. It must also be stressed that any massive connection to the values of the parameters should be avoided because it could overload the event manager.

The solution to these issues is the introduction of histograms for displaying the distribution of the values for all kinds of parameters. Histograms are widely used in the scientific environments and so the final

[1]For the DCUs with up to 15 000 values simultaneously

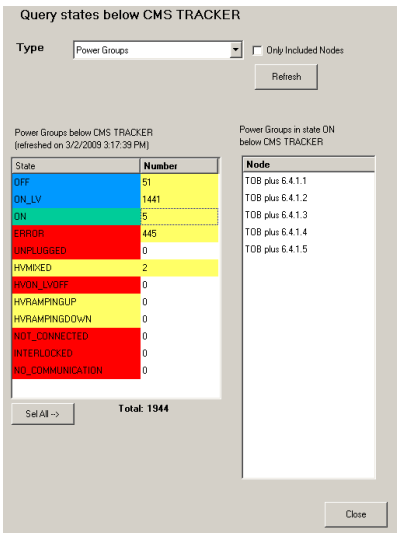

Figure 4.34: The interface to query the state of all the nodes of a given type below a certain node. The list of nodes in a certain state can be easily displayed. With a further click, the detailed panel of a single node can be found.

users of the control systems are very familiar with the tool. The histogram is built by getting the parameters on demand, so that a connection to the changing values is not needed. To enable the search of devices with a specific value, the bins of the histogram can be clicked, to obtain the list of all the devices whose value is included in the corresponding interval. Clicking again, the details of the individual device can be displayed. This approach is very effective to find devices where a parameter has an unusual value. In addition, all the retrieved data can be saved to a file that can be imported in any external analysis tool.

A similar approach is used for searching the nodes in a given state. In an interactive interface (see Fig. 4.34), the user can choose the type of node of interest (e.g., cooling loop, control group, power group) and get all the states of the nodes of that type in a specific subtree. The number of nodes in each state is displayed in a table and the list of nodes in a given state can then be retrieved.

The "get on demand and compute" approach is also used to implement specific expert tools, such as the computing of the power in the racks and the efficiency of the power supply system.

4.17.3 Access Control

The definition of the access control for the CMS tracker control system uses the standard framework access control described in Section 3.4.7.

Three roles corresponding to different levels of expertise are defined. The standard shifter can switch the tracker on and off using the Finite State Machine commands but cannot change the configuration of the power supplies. The detector expert can change some power supply parameters (inside some defined limits), while the DCS expert is allowed to change some general software configurations (for example, the interlock thresholds in the PLC).

A separate domain and role is used for the control of the cooling system, since the experts in this field are not necessarily detector or DCS experts.

The access control defines the appearance of the GUI. More information is displayed to the expert user, while the standard shifter is presented only with the essential information.

4.17.4 Tasks and User Interaction During a Typical Shift

During a typical shift, the operators use the DCS system to control and monitor the detector. When there are no major problems in the tracker, user interaction is very limited. At the beginning of the run, the DCS shifter gives the command ON to the entire tracker or to the TTC partitions that need to be operated and waits for the feedback that the power supply system is completely on. The complete ramping up phase can last some minutes. All the trips in the channels typically happen during the switching-on phase. The errors in the channels are reported both in the Alarm Screen and in the percentages table. Both tools allow the user to click and find immediately the channel(s) involved. Usually a second trial is sufficient to bring these channels to operation. Otherwise the intervention of an expert to tune some parameters (set voltage, current limit, etc.) may be needed.

A workstation with at least two screens is used for the operation of the DCS interface (eventually four screens will be used). A predefined set of trending panels is kept open to display the evolution of the system parameters in real time. Standard trends include the minimum and maximum temperatures in the four TTC partitions (to immediately detect a temperature increase), the mean temperature in all cooling loops and sectors, the cooling plant tank levels and the trend of the dew point values.

The shifter actions are never critical for detector safety. The access control prevents the shifter from applying dangerous settings to the hardware systems. Any safety action is carried out automatically by the DCS software, and, if the DCS action fails, by the hardware Tracker Safety

System. In case of a critical event, an audio alarm signals the condition to the user. The recovery from a critical condition is possible via the Tracker Wizard (see Sec. 4.14) that is able to recover from most situations or to identify the cause of the problem and give the name of the proper expert to call.

The searching tools provided by the propagation algorithm or by the Alarm Screen allow for an easy identification of the problematic devices. These tools are much more effective than any FSM-based hierarchy browsing and were a key point for increasing the efficiency of routine tasks. Moreover, the automation of repetitive operations in a task carried out by a wizard minimizes the level of expertise needed by the shifters. During normal operation, the tracker could be successfully operated by shifters who followed a short introduction course (less than one hour) on the usage of the DCS interface.

Chapter 5

CMS Tracker as a Case Study for Automatic 3D GUI Prototyping for Control Systems

This chapter presents a preliminary study investigating the possible use of a model-based methodology for automatically building a three-dimensional user interface for the CMS Tracker Control System. The study shows that the proposed methodology can be successfully applied to a real-world case study. The problematic aspects that still have to be addressed in order to extend the methodology to a large scale control system are identified in this study.

5.1 Introduction and General Objectives

Building three-dimensional interfaces for control systems is a complex task that requires good knowledge of the modeled domain and involves many issues relating to user interaction and the development process.

For typical computer displays, "three-dimensional graphical user interface" is actually a misnomer since their displays are two-dimensional projections of three-dimensional images. Although some stereoscopy experiments tried to reproduce an illusion of depth in the image, a real three-dimensional user interface is still unavailable.

Three-dimensional representation of complex devices can help a human operator to find spatial correlations of errors or states of the hardware. For example, in the case of LHC detectors, unstable particle beams in the accelerator could produce several errors (voltage trips, etc.) in various parts of the sub-detectors. Identifying this beam problem in a

classical 2D or hierarchical view may be difficult. Instead, a 3D view can show that alerts are located around a certain region of the detector allowing a faster identification of the problem.

Even so, a three-dimensional interface is not an absolute must for a control system. Traditional interfaces can implement all the fundamental features without an advanced geometrical visualization of the controlled devices. However, to guide the future developments, it is interesting to investigate how the classical hierarchical approach can be merged and integrated with a more user-oriented three-dimensional interface.

The approach to the problem presented in this chapter is a formal methodology for automatic prototttyping 3D user interfaces for control systems. The BATIC^{3}S (Building Adaptive Three-dimensional Interfaces for Critical Complex Control Systems) project is an international collaboration involving various universities and research institutes (*Université de Genève*, *Universidade Nova de Lisboa*, *Ecole d'ingénieurs de Genève*, *HES-SO Valais* and CERN). Its goal is to introduce a methodology to produce a GUI from a formal specification of the characteristics of the system under control and to develop an effective framework that implements the methodology. The CMS tracker is used as a case study to investigate the possibilities of applying this methodology to a real-word example and to test the possibility of interfacing the prototypes to a SCADA system for the control of a real physical apparatus.

5.2 The BATIC^{3}S Methodology and Framework

5.2.1 Introduction

The project proposes a methodology to develop 3D graphical user interfaces for monitoring and controlling complex control systems [38]. The interface is not directly specified nor developed, but is automatically prototyped from the knowledge about the system under control.

Since the methodology and the implementation of the framework is not the subject of the present thesis, only the basic concepts are presented here. All the details can be found in the bibliography.

Modeling the domain of control systems has requirements and challenges which are hardly met by standard, general-purpose modeling languages. A domain specific language for specifying control systems has therefore been designed as a part of the project.

The implementation of the methodology in a coherent framework has been developed by integrating various well-known and open tools.

5.2.2 Domain Definition and Requirements

The domain of Control Systems addressed in the context of the project is completely analogous to the scope of the detector control systems for high energy physics experiments. A Control System is defined as a mechanism to provide output variables of a system by manipulating its inputs (from sensors or commands). A control system generally has a composite structure, in which objects can be grouped with others, forming a hierarchical tree where the root represents the whole system and the leaves are its most elementary devices. This hierarchical approach is compatible with (and actually inspired by) the FSM partitioning approach.

The aim of the project is to provide the system expert with the tools to specify a model, based only on the knowledge of the system under control, without a direct GUI programming. From the specification in the domain specific language it must be possible to generate an executable prototype of the GUI, verify properties to validate the specification, classify users into profiles and define tasks which may be available to specific user profiles.

5.2.3 Methodology

The engineering process for prototyping a GUI is split into four steps, as illustrated in Fig. 5.1. In the first step, the knowledge about the system is gathered from existing documentation or from human experts. In the second step, this information is expressed using a domain specific language. The specification is composed of four distinct models. The *system model* describes the structure and the behavior of the system under control. The *visual model* describes the geometry of the constituting objects. The *user model* and the *task model* describe the interactive aspects of the system. The model is not GUI-oriented.

In the third step, the system model is automatically transformed into two independent components: a visual and interaction model, which is stored in a database and contains all information about the geometry, and an executable system simulator. The simulator has a formally defined semantics and uses the CO-OPN (Concurrent Object Oriented Petri Net) [39] language.

The fourth and final step is the dynamic generation of a GUI prototype built from the database, which interacts with the system simulator. Finally, the simulator can be replaced by a driver communicating with the real system. The simulator is useful because for various reasons (cost, time, unavailability of hardware) the real system may not be available for evaluating the GUI against the real reactive system.

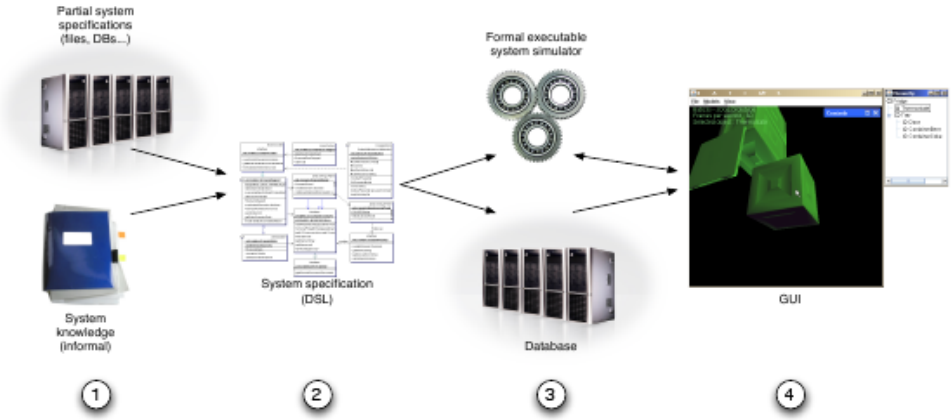

Figure 5.1: Schematic view of the methodology process. A system specification is created by gathering information from existing files or databases or from informal system knowledge. Two components are automatically created from the system specification: a system simulator and a geometrical database. The GUI loads the system specification from the database and presents the user with a 3D interface. The interface uses the system simulator to provide a feedback to the user. Finally the system simulator can be replaced with a driver providing the communication with the real system.

5.2.4 The Domain Model

The **system model** describes the logical structure of the system. The concepts used for modeling are: *objects*, *types*, *hierarchical composition*, *object behavior*, *state dependency*, *properties*, *commands* and *events*.

Objects represent components in the system. Each object is of a specific *type*. All features that are not specific to individual instances of the objects are defined in the type.

The *hierarchical composition* is defined as a tree of objects. The *behavior* of the object is modeled by Finite State Machines. The *state dependency* is expressed with conditional rules on the children objects in the hierarchical tree. *Properties* are specific values that can be attached to objects. *Commands* define an action on an object. For the sake of simulation, a simplified behavior for the commands on the devices can be provided. *Events* are triggered by state transitions (also of the children objects in the hierarchical tree), commands, or property changes.

The behavior of the objects, their commands, events, and properties are associated to the types, because these features are common to all objects of a given type.

The **visual model** describes the geometry of the devices. It is composed of the *geometrical shape* information, that is related to the type, and the *position in space*, that is related to each object. The definition of the geometrical shape can make use of predefined common primitive shapes (box, sphere, cylinder...) or load a user-defined geometry file. The position can be specified in absolute coordinates (translation and rotation) or in relation to another object, in order to ease the definition of repetitive patterns in the geometry.

The **user model** is based on a subset of the *Generic Ontology based User Model* [40]. Each user (or *agent*) is mapped to a profile (*behavior*) and to the knowledge of a specific task, with a certain expertise level.

The **task model** defines how the user can reach a goal in a specific application domain. The task model is based on the ConcurTaskTrees formalism [41]. According to this formalism, tasks can be classified into four types: abstract (a complex task defined in terms of its subtasks), user (that is something that the user does without interaction with the system, e.g., a decision), application (something that is executed by the system) and interaction (e.g., clicking a button).

Tasks can be combined by means of process algebra operators. For example, `T1 ||| T2` means interleaving (the actions of the two tasks can be performed in any order), `T1 [] T2` represents a choice between two different tasks and `T1 » T2` is an enabling relation, meaning that the second task is activated when the first one is terminated.

The domain model specification is implemented in a custom designed Domain Specific Language (DSL), called COntrol system SPEcification Language (Cospel), that is made of different packages, modeling the different aspects of the system. The abstract syntax of Cospel is specified using the Eclipse Modeling Framework (EMF). Thanks to EMF, a visual editor is automatically generated from the abstract syntax of the language and can be used by the system engineer for specifying the system model.

Properties of the structure of the system can be directly checked in the specification using a constraint language on the model. It is possible for example, to check if a certain type of object always has children of a certain other type.

5.2.5 System Simulator and Model Transformation Techniques

The system simulator is obtained through model transformation techniques from the syntactic specification by giving executable semantics to the model.

The result is an executable concurrent model that implements the execution of the FSMs, the command input model and the event model.

The semantics of a Cospel model is obtained by defining proper mapping rules between the Cospel metamodel and the CO-OPN metamodel. CO-OPN (Concurrent Object-Oriented Petri Nets) is an object oriented modeling language based on algebraic Petri Nets, providing tools for simulation, verification and test generation.

The model transformation is performed with the ATLAS Transformation Language (ATL) [42], a declarative language in which rules can be defined to transform an input model, conforming to a given metamodel, to an output model, conforming to another metamodel. In this way, since the semantics of CO-OPN is well-defined, the transformation gives semantics to the Cospel syntax.

ATL produces another model from the Cospel specification, that is used to fill the geometry database. Since the system simulation is purely behavioral, no visualization-related information is used for building the CO-OPN model.

From the point of view of the user, the model transformation is an automatic one-click operation because the transformation rules have been defined in the framework,.

Properties and constraints of the model can be specified in a suitable formalism, like temporal logic, and can be automatically verified by a model checking tool on the state space of the simulator. Testing is also

possible: the simulator is fed a known input and the is output checked for the expected values.

5.2.6 GUI Prototype

The GUI engine is capable of loading the system specification from the database and presenting the user with an interface allowing interaction with the system. A driver instantiates and runs the system simulator, that emulates the feedback of the real system. Commands and events are transmitted to and from the driver. The driver can instantiate the system simulator or provide the communication with the actual controlled system.

The task model information is used to show the available tasks for a given object. When the task is purely sequential (composed only of enabling operators), it can be completely automated in the GUI. Otherwise a wizard-like approach can be adopted presenting the user with the possible choices. The user model is used as an authorization model for the determination of the available tasks.

A crucial problem in the implementation of three-dimensional GUIs is the definition of a strategy to highlight components that require special attention from the user (for example, in case of error). In a three-dimensional environment, an object may be hidden by other objects or may be out of the current view. To address this problem, without forcing the user to specify low-level adaptation behavior in the specification of the model, the GUI engine uses *adaptation rules*. These rules depend on the user profile and on current tasks and are triggered by an error event. Common strategies used in this context are: centering the camera on the object, moving objects which block the view of the faulty object or making them transparent.

The GUI engine provides both a three-dimensional view built according to the geometry and a hierarchical view that shows the control tree of the system under control. The two views correspond to two different navigation modes, spatial and hierarchical, that are complementary and can be both useful for different types of tasks. The spatial navigation is useful to find spatial correlation when investigating faults and alerts. When alerts are of environmental nature, it makes sense to check the geometrical nearby components. The spatial navigation system provides easy access to the nearby objects by moving the camera and clicking on them. This also works well for components that might belong to a whole different branch in the system hierarchy. For example, in the case of the tracker, the outer layer of the TIB is spatially adjacent to the inner layer of the TOB but this proximity is not reflected in the control hierarchy.

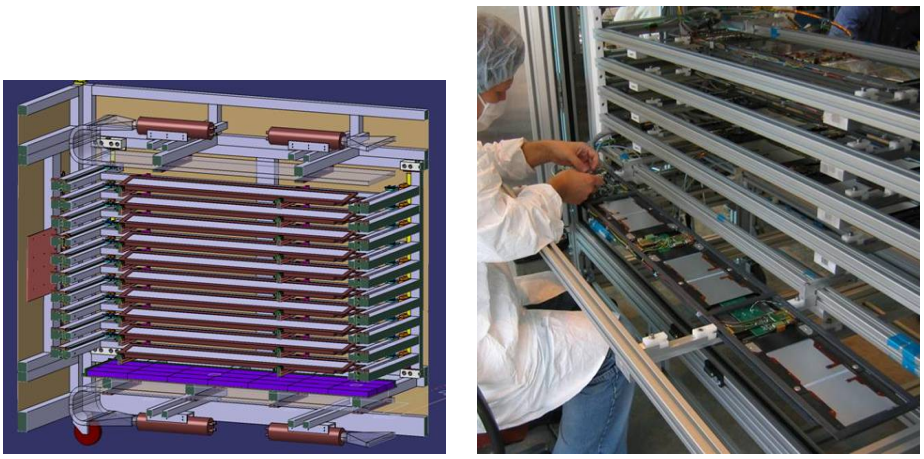

Figure 5.2: Schematic view of the Cosmic Rack and a picture of the Cosmic Rack during the integration phase

On the other hand, if the user wants to quickly jump to a specific part of the system, spatial navigation is not efficient as it requires several zooming, panning and rotating operations. In these cases, the hierarchical view provides a quick focus switch to any region of the system.

Rendering is done by the JoGL API, a Java binding of the OpenGL libraries. JoGL has built-in support for stereoscopic simulation.

The GUI can be used for validation purposes: tasks can be automatically executed to check if they have the expected behavior, or the validation can be performed in a more user-centric way, by asking the user to execute a certain task and evaluating manually the ease (or possibility) to perform it.

According to the common practice in the field, the states of the objects are represented in the scene by colors (green=ok or on, red=error, etc). When an object is clicked, additional information about its state is displayed with a string description (OFF, ON, MIXED, etc). When an object is selected and if the current user model has the authorization to perform a task (according to the user model specified in the model), the commands are displayed in the interface.

5.3 The Cosmic Rack Case Study

The CMS Tracker Cosmic Rack has been chosen as a case study [43] for the application of the BATIC^{3}S methodology. The Cosmic Rack is a structure used to make Data Acquisition tests before the complete tracker was assembled. It corresponds to a section of the TOB and, from the point of view of control, maintains the same hierarchical complexity as

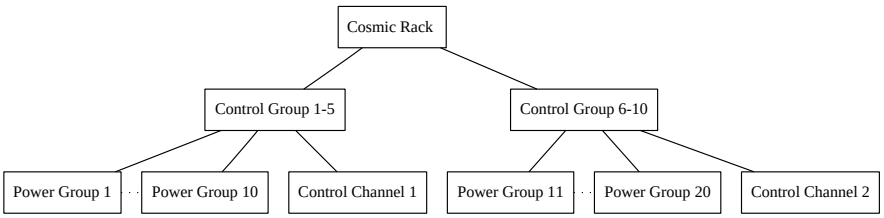

Figure 5.3: Cosmic Rack control hierarchy

the tracker, but with a reduced number of components. The test setup corresponds to a cooling segment (cooling loop) and has two control rings and a total of 20 rods (power groups) of 6 modules each.

In the Cospel specification, four types are defined (power group, control channel, control group, cooling loop). The state and the logic of these types are completely analogous to the CU and DU types implemented in the JCOP FSM for the control of the tracker (see Sec. 4.2). The rules define the state depending on the state of the children and without taking into account the percentages. The types are instantiated in the tree represented in Fig. 5.3.

The geometry of the cosmic rack is specified in the model on the basis of an informal description. The cosmic rack hosts 20 rods (each rod is a power group) organized in 10 layers. Each control group serves 5 layers (see Fig. 5.2). If the case study has to be extended to the whole tracker, a more precise geometry description will have to be retrieved from an existing geometry database. In any case, both the functional and the geometrical description used for modeling are normally already available in the specification of a complex control system.

5.4 Outcomes of the Collaboration and Further Development

The collaboration aimed to be useful for both the CMS Tracker Control System and the BATIC^{3}S collaboration. From the point of view of the BATIC^{3}S project, an important point to be investigated is the possibility to connect the GUI engine with a real SCADA system. In fact, even if the architecture of the system is designed to allow the system simulator to be easily replaced by an external system, in practice this test had never been performed previously. In order to establish the connection to a PVSS project controlling the Cosmic Rack, the obvious choice is to use PSX, a server that allows any external application to communicate

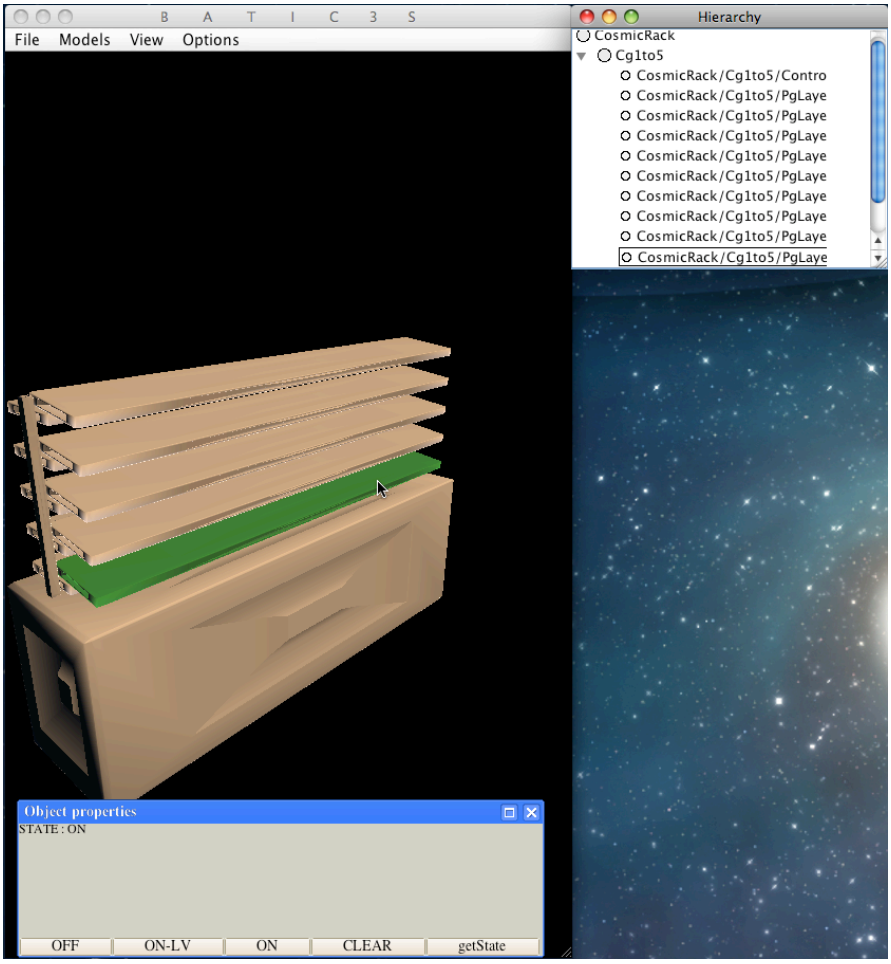

Figure 5.4: Screenshot of the prototyped 3D user interface for the control of the Cosmic Rack. One rod is selected. The state of all the rods in control group "1 to 5" and the state of control group "6 to 10" are shown in the interface

to PVSS via the standard SOAP protocol. An existing interface for PSX written in Java (providing a high level of abstraction) is used to implement the driver. PSX provides the means to connect to the state of an FSM object and to send commands directly to the JCOP FSM. In a first phase, the GUI was connected to a PVSS system simulating the feedback for the power supply channels with a script. Finally the interface was tested on the real Cosmic Rack hardware.

From the point of view of the Tracker Control System, the use-case was useful to investigate how a 3D visualization can simplify the user interaction and to determine if the three dimensional interface is effective enough for operation or must be integrated with the standard FSM interface. To be able to extend the same methodology to the entire tracker, the model specification has to be automated by converting existing structural and geometrical information to the proper format.

Moreover, the exercise underlined various problematic aspects that have to be addressed:

- As discussed in Section 4.17.2, it is important to avoid the connection to the status of all the controlled objects (see Sec. 4.17.2). In fact, in case of a control system dealing with a large number of objects, this strategy could overload the Event Manager. Moreover, in the case of the tracker, the outer layers and disks are hiding the inner parts, making difficult to visualize the state of all the equipment (this is also true for the cosmic rack, even if in this case the situation is less severe because only a section is represented). The FSM states offer a higher level of abstraction and should be somehow exploited also in a three-dimensional interface. A possible solution is to connect to the states of some higher level objects (e.g., a control group) and only connect to the lower level objects when a detailed view of a certain region is needed.

- The geometry for the higher level objects (such as a control group) that do not have a clear physical counterpart has to be somehow defined. One possibility, that can be appropriate in case of non overlapping regions, is to use the bounding box of the contained elementary objects. Otherwise one can define the geometry of a high level object as the union of all the geometries of the contained elementary objects (treated as a single object and hence filled in the same color).

- The 3D interface cannot replicate all the functionalities of the standard DCS GUI. Therefore, the Java interface must be integrated

with the PVSS GUI in order to be able to use the prototyped interface in a real control system. For example, it should be possible for the user to click on the 3D representation of the object and open the related PVSS panel.

Conclusions

This thesis gives a detailed overview of the problems arising in the implementation of control systems for large scale and complex equipment, such as the LHC experiments.

The CMS Silicon Strip Tracker, the largest silicon tracker ever built and one of the most complex CMS sub-detectors from the point of view of control, has been used as a real-world example for verifying the validity of the proposed solutions.

While the DCS for a HEP experiment is built on top of a SCADA product used in industrial automation, it has some HEP specific requirements that call for extended functionalities. Moreover, the complexity of the control tasks for operating the tracker call for innovative solutions with respect to the general framework developed at CERN for the control of the LHC experiments.

My work includes detailed performance studies of the SCADA product chosen for implementing the control systems of the LHC experiments and of the drivers used in the communication with the devices. The results of these studies led to a fine tuning of the communication with the hardware and to some architectural choices, such as caching of the statical information and maximal role distribution among the control PCs, to ensure an acceptable performance.

An especially challenging issue is the effective representation of the state of a system composed of a large number of parameters. A hierarchical approach is obviously needed, but a single state is often not enough to cover all the relevant aspects. The propagation algorithm presented in this work provides a flexible and efficient tool for integrating the control hierarchy with quantitative information. It is now a standard tool supported centrally for the entire CMS experiment and has been adopted by other CMS sub-detectors, such as RPC and CSC.

Additionally, the standard hierarchical approach is not suited for ensuring that the detector is in a safe state (e.g., all the HV channels are off before beam injection). Rather than using the standard control chain, the lowest level is checked and any command exiting from the safe state is inhibited. The package implementing these requirements is now used

for all CMS sub-detectors.

Although the detector safety is ensured by independent hardware systems, DCS is responsible for their configuration. The proper configuration of the Tracker Safety System is cross-checked by comparing the real hardware behavior with the data stored in an *ad hoc* configuration database.

A HEP experiment must be controlled and monitored by non expert operators that should be able to perform most of the needed tasks, including error recovery, without expert intervention. A usable and responsive user interface is a key component of a control system. Common operator tasks are simplified by using an efficient search mechanism to easily identify devices in error or with unusual values. A wizard approach to the resolution of problems allows non expert operators to recover problematic situations that require the analysis of numerous parameters.

In addition, the possibility of using three-dimensional user interfaces is investigated. This kind of UI may be very useful to to find spatial correlations of errors or states of the hardware. The model-based approach presented in the last chapter allows for automatic prototyping of 3D user interfaces.

The control system components are still open to substantial improvement, especially in the communication layer where better performance in the communication with the hardware devices can be achieved. At the supervisory layer, the systematic use of an object oriented methodology may improve the maintainability and extensibility of SCADA systems.

As the issues presented in this thesis show, the state-of-the-art software solutions commonly employed in automation industry are not ideally suited to the number of devices and complex requirements of a large HEP experiment. However, they can still successfully be used in this context with the help of special techniques presented in this thesis.

Acknowledgements

It is a pleasure to thank those who made this thesis possible, first of all my supervising professor, Prof. Volker Lindenstruth, whose support during thesis writing greatly helped me in the presentation of my work. I would also like to thank Prof. Dr. Peter Fischer for taking time from his busy schedule to serve as my second referee. I am particularly grateful to Dr. Jan de Cuveland for his punctual remarks and advices that helped me finalizing the thesis.

I wish to thank all the people of the CMS Tracker DCS and DSS team I have worked with during the last four years: Frank Hartmann, Guido Dirkes, Andromachi Tsirou, Piero Giorgio Verdini, Manuel Fahrer, Yousaf Shah, Robert Stringer, Otilia Militaru, Martin Frey, William Gabella, Helena Malbouisson, Christian Barth. Most of the original solutions and strategies presented in this thesis were designed, discussed and reviewed during long and enlightening meetings within this group. Frank Glege and Robert Gomez-Reino Garrido from the central DCS group also contributed to the design of some key components, in particular the *Detector Protection* package.

A special thanks to Verena for her precious help in making my English clearer and more understandable.

Oliver Holme has made available his support in a number of ways, helping the Tracker DCS team to make the right choices during the design phase and providing feedback for the performance studies on PVSS.

Some DCS developers of other CMS sub-detectors particularly helped me in the generalization of the *majority* and *Detector Protection* packages. Thanks to Marina Giunta, Diogo Di Calafiori, Giovanni Polese, Evaldas Juska.

I would also like to thank all people from the CMS group in Florence, where I started my work on DCS. I am particularly grateful to professor Gregorio Landi for introducing me to work in this field, and to Simone Paoletti for his advice and help with the present thesis.

It was a pleasure to work with Matteo Risoldi and Prof. Didier Buchs from the University of Geneva at the model-based studies used for 3D UI prototyping.

Bibliography

[1] A. Daneels and W. Salter. What is SCADA? In *proceedings of the 7th International Conference on Accelerator and Large Experimental Physics Control Systems, Trieste, Italy*, 1999.

[2] S. Bimbo and E. Colaiacovo. *Sistemi SCADA Supervisory control and data acquisition*. Apogeo online, 2006. available online at http://www.apogeonline.com/libri/88-503-1042-0/scheda.

[3] L. Wang and K. C. Tan. *Modern Industrial Automation Software Design*. Wiley, Newark, NJ, 2006.

[4] OPC fundation. OPC specifications. available online at http://www.opcfoundation.org.

[5] Ronald L Krutz. *Securing SCADA Systems*. Wiley, Newark, NJ, 2006.

[6] S. Lüders. Summary of the Control System Cyber-Security (CS)2/HEP Workshop. In *proceedings of the 11th International Conference on Accelerator and Large Experimental Physics Control Systems, Knoxville, TN, USA*, 2007.

[7] S. D. Garbrecht. The Benefits of Component Object-Based SCADA and Supervisory System Application Development. available online at http://global.wonderware.com/EN/PDF%20Library/-COBSS_wp_Final_r1_a.pdf.

[8] T. S. Pettersson and P. Lefèvre. The Large Hadron Collider: conceptual design. Technical Report CERN-AC-95-05 LHC, CERN, Geneva, Oct 1995.

[9] *CMS, the Compact Muon Solenoid: technical proposal*. LHC Tech. Proposal.

[10] S M Sze. *Physics of semiconductor devices*. Wiley, New York, NY, second edition, 1981.

[11] F. Hartmann. *Evolution of silicon sensor technology in particle physics*. Springer Tracts in Modern Physics. Springer, 2008.

[12] S. Paoletti, A. Bocci, R. D'Alessandro, and G. Parrini. The Powering Scheme of the CMS Silicon Strip Tracker. In *Proceedings of 10th Workshop on Electronics for LHC and Future Experiments, Boston*, 2004.

[13] S. Paoletti. The Implementation of the power supply system of the CMS Silicon Strip Tracker. In *Proceedings of the Topical workshop on Electronics for Particle Physics, TWEPP-07, CERN report CERN-2007-007, pp 377-81*, 2007.

[14] E. Carrone, A. Tsirou, and P. G. Verdini. A discrete event system for the CMS Tracker interlocks. In *proceedings of 10th International Conference on Accelerator and Large Experimental Physics Control Systems, Geneva, Switzerland*, 2005.

[15] G. Magazzù, A. Marchioro, and P. Moreira. The Detector Control Unit: An ASIC for the monitoring of the CMS silicon tracker. *IEEE Trans. Nucl. Sci.*, 51:1333–1336, 2004.

[16] A. J. Bell. Beam & Radiation Monitoring for CMS. In *proceedings of the Nuclear Science Symposium, Medical Imaging Conference and 16th Room Temperature Semiconductor Detector Workshop, Dresden, Germany*, 2008.

[17] M. Bellato, L. Berti, V. Brigljevic, G. Bruno, E. Cano, S. Cittolin, A. Csilling, S. Erhan, D. Gigi, F. Glege, et al. Run Control and Monitor System for the CMS Experiment. In *proceedings of Computing in High Energy and Nuclear Physics 2003 conference, La Jolla CA*, 2003.

[18] S. Schmeling, R. Bruce Flockhart, S. Lüders, and G. Morpurgo. The detector safety system for LHC experiments. *IEEE Trans. Nucl. Sci.*, 51:521–525, 2004.

[19] A. Daneelsand and W. Salter. Selection and Evaluation of Commercial SCADA Systems for the Controls of the CERN LHC Experiments. In *International Conference on Accelerator and Large Experimental Physics Control Systems, Trieste, Italy*, 1999.

[20] P. C. Burkimsher. JCOP Experience with a Commercial SCADA Product, PVSS. In *International Conference on Accelerator and Large Experimental Physics Control Systems, Gyeongju, Korea*, 2003.

[21] P. C. Burkimsher. Scaling up PVSS. Technical Report CERN-OPEN-2005-029, CERN, Geneva, Oct 2005.

[22] JCOP Architecture Working Group. Framework Design Proposal. Technical Report CERN-JCOP-2000-008, CERN, Geneva, 2001.

[23] B. Franek and C. Gaspar. SMI++ Object Oriented Framework for Designing and Implementing Distributed Control Systems. Technical Report SLAC-PUB-12067, SLAC, Stanford, CA, Sep 2006.

[24] C. Gaspar. Hierarchical Controls Configuration & Operation. Technical report, CERN, 2004. available online at http://lhcb-online.web.cern.ch/lhcb-online/ecs/fw/FSMConfig.pdf.

[25] M. González-Berges. The high-performance database archiver for the LHC experiments. Technical Report CERN-IT-Note-2007-027, CERN, Geneva, Oct 2007.

[26] S. Lüders. Update on the CERN Computing and Network Infrastructure for Controls (CNIC). Technical Report CERN-IT-Note-2007-022, CERN, Geneva, Sep 2007.

[27] F. Varela. Software management of the LHC detector control systems. In *proceedings of the 11th International Conference on Accelerator and Large Experimental Physics Control Systems, Knoxville, TN, USA*, 2007.

[28] CMS DCS Integration Guidelines document.

[29] M. Frey on behalf of the CMS Tracker Collaboration. The CMS Tracker Detector Controls System. 2005.

[30] F. Hartmann for the CMS Tracker Control System Group. The CMS Tracker Control & Safety System. In *proceedings of the Vienna Conference on Instrumentation VCI 2007*, 2007.

[31] A. Dierlamm, G. H. Dirkes, M. Fahrer, M. Frey, F. Hartmann, L. Masetti, O. Militaru, S. Y. Shah, R. Stringer, and A. Tsirou. The CMS tracker control system. *Journal of Physics: Conference Series*, 119(2):022019 (9pp), 2008.

[32] M. Fahrer, J. Chen, A. Dierlamm, M. Frey, F. Hartmann, L. Masetti, O. Militaru, S. Y. Shah, R. Stringer, and A. Tsirou. The Control System for the CMS Strip Tracking Detector. In *proceedings of 10th ICATPP Conference on Astroparticle, Particle, Space Physics, Detectors and Medical Physics Applications, Villa Olmo, Como, Italy*, 2007.

[33] L. Masetti, F. Hartmann, S. Y. Shah, and R. Stringer. The CMS Tracker Detector Control System. In *proceedings of the Nuclear Science Symposium, Medical Imaging Conference and 16th Room Temperature Semiconductor Detector Workshop, Dresden, Germany*, 2008.

[34] S. Y. Shah, A. Tsirou, P. G. Verdini, F. Hartmann, L. Masetti, G. H. Dirkes, R. Stringer, and M. Fahrer. The CMS tracker detector control system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, In Press, Corrected Proof, 2009.

[35] A. Barriuso-Poy. *Hierarchical Control of the ATLAS Experiment*. PhD thesis, Universitat Rovira i Virgili, Departament d'Enginyeria Electrònica, Elèctrica i Automàtica, Tarragona, 2007. Presented on 14 May 2007.

[36] P. Milenovic, A. Brett, G. Dissertori, G. Leshev, T. Punz, S. Zelepoukine, S.D. Di Calafiori, R. Gomez-Reino, R. Ofierzynski, et al. The Detector Control System for the Electromagnetic Calorimeter of the CMS Experiment at LHC. In *proceedings of the 11th International Conference on Accelerator and Large Experimental Physics Control Systems, Knoxville, TN, USA*, 2007.

[37] B M González Berges, F Varela, and K Joshi. The system overview tool of the joint controls project (jcop) framework. In *proceedings of the 11th International Conference on Accelerator and Large Experimental Physics Control Systems, Knoxville, TN, USA*, 2007.

[38] V. Amaral, B. Barroca, M. Risoldi, D. Buchs, and G. Falquet. *A language and a methodology for prototyping user interfaces for control systems*, pages 223–252. Springer-Verlag, 2009.

[39] O. Biberstein. *CO-OPN/2: A concurrent object-oriented formalism for the Specification of Concurrent Systems*. PhD thesis, University of Geneva, 1997.

[40] A.L. Calvé F. Cretton. Generic ontology based user modeling - GenOUM. Technical report, HES-SO Valais, Sierre, Switzerland, 2007.

[41] F. Paternò, C. Mancini, and S. Meniconi. ConcurTaskTrees: A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 Interantional Conference on Human-Computer*

Interaction, pages 362–369. Chapman & Hall, Ltd. London, UK, UK, 1997.

[42] ATLAS Group. Atlas transformation language, 2008. http://www.eclipse.org/m2m/atl/.

[43] M. Risoldi, D. Buchs, L. Masetti, V. Amaral, and B. Barroca. A Methodology for Control Systems GUI Prototyping - a case study. In *7th international workshop on Personal Computers and Particle Accelerator Controls (PCaPAC 2008), Ljubljana, Slovenia*, 2008.